FPGA Based Network Security architecture for High Speed Networks

Thesis submitted in partial fulfillment of the requirements for the degree of

Master of Technology

in

Computer Science and Engineering

(Specialization: Information Security)

by

Sourav Mukherjee

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela, Odisha, 769 008, India

May 2011

FPGA Based Network Security architecture for High Speed Networks

Thesis submitted in partial fulfillment of the requirements for the degree of

Master of Technology

in

Computer Science and Engineering

(Specialization: Information Security)

by

Sourav Mukherjee

(Roll- 209CS2090)

Supervisor

Prof. Bibhudatta Sahoo

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela, Odisha, 769 008, India

May 2011

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela-769 008, Odisha, India.

Certificate

This is to certify that the work in the thesis entitled ***FPGA based Network Security Architecture for High Speed Networks*** by ***Sourav Mukherjee*** is a record of an original research work carried out by him under my supervision and guidance in partial fulfillment of the requirements for the award of the degree of Master of Technology with the specialization of Information Security in the department of Computer Science and Engineering, National Institute of Technology Rourkela. Neither this thesis nor any part of it has been submitted for any degree or academic award elsewhere.

Place: NIT Rourkela
Date: 30 May 2011

Bibhudatta Sahoo
Professor, CSE Department
NIT Rourkela, Odisha

Acknowledgment

It has been a long journey for me when I started this work and now when I am writing this, I am going to take the privilege of thanking those people who assisted me a lot for completing my work .

First of all, I would like to express my sincere thanks to Prof. Bibhudatta Sahoo for his advice during my thesis work. As my supervisor, he has constantly encouraged me to remain focused on achieving my goal. His observations and comments helped me to establish the overall direction of the research and to move forward with investigation in depth. He has helped me greatly and been a source of knowledge.

I am very much indebted to Prof. Ashok Kumar Turuk, Head of the Department, CSE, for his continuous encouragement and support.I am also thankful to Dr. B. Majhi, Dr. S. K. Rath, Dr. S. K. Jena, Dr. D. P.Mohapatra, Dr. R. Baliarsingh, and Dr. P. M. Khilar for giving encouragement and sharing their knowledge during my thesis work.

I am really thankful to my all friends and my lab mates and specially the seniors of the VLSI Lab in the ECE department, who helped me every time when I was in some trouble.

I must acknowledge the academic resources that I have got from NIT Rourkela. I would like to thank administrative and technical staff members of the Department who have been kind enough to advise and help in their respective roles.

Last, but not the least, I would like to dedicate this thesis to my parents and my sister, for their love, patience, and understanding.

Sourav Mukherjee

Email ID: souravaot@gmail.com

Abstract

Cryptography and Network Security in high speed networks demands for specialized hardware in order to match up with the network speed. These hardware modules are being realized using reconfigurable FPGA technology to support heavy computation. Our work is mainly based on designing an efficient architecture for a cryptographic module and a network intrusion detection system for a high speed network. All the designs are coded using VHDL and are synthesized using Xilinx ISE for verifying their functionality. Virtex II pro FPGA is chosen as the target device for realization of the proposed design. In the cryptographic module, International Data Encryption Algorithm (IDEA), a symmetric key block cipher is chosen as the algorithm for implementation. The design goal is to increase the data conversion rate i.e the throughput to a substantial value so that the design can be used as a cryptographic coprocessor in high speed network applications. We have proposed a new n bit multiplier in the design which generates less number of partial products ($\leq \frac{n}{2}$) and the operands are in diminished-one representation. The multiplication is based on Radix-8 Booth's recoding with different combinations of outer round and inner round pipelining approach and a substantial high throughput to area ratio is achieved. The Network Intrusion Detection System (NIDS) module is designed for scanning suspicious patterns in data packets incoming to the network. Scanning a data packet against multiple patterns in quick time is a highly computational intensive task. A string matching module is realized using a memory efficient multi hashing data structure called Bloom Filter, in which multiple patterns can be matched in a single clock cycle. A separate parallel hash module is also designed for eliminating the packets which are treated as false positives. The string matching module is coded and functionally verified using VHDL targeting Virtex II pro FPGA and performance evaluation is made in terms of speed and resource utilization.

Contents

Certificate ii

Acknowledgement iii

Abstract iv

List of Figures vii

List of Tables viii

1 Introduction 2

1.1 Motivation 3

1.2 Problem Statement 4

1.3 Our Contribution 5

1.4 Thesis Organization 6

2 Background 8

2.1 Introduction 8

2.2 Symmetric key Cryptosystem 8

2.2.1 IDEA Encryption Algorithm 8

2.3 Related Work 10

2.4 Conclusion 18

3 Modulo $(2^{16}+1)$ multiplier for IDEA Cipher 20

3.1 Introduction 20

3.2 Diminished-one Number Representation 20

3.2.1 Basic Operations 21

3.3 Algorithm for the proposed multiplier 21

3.4 Proposed multiplier architecture 26

3.5 Complexity of the proposed multiplier : 27
3.6 Results and Analysis 28
3.7 Conclusion 29

4 Design and Implementation of IDEA cipher 31

4.1 Introduction 31
4.2 Design and Architecture using pipelining 31
4.3 Result and comparison with other schemes 34
4.3.1 Analysis and Comparison 34
4.4 Conclusion 36

5 FPGA based string matching for Network Intrusion Detection System 39

5.1 Introduction 39
5.2 Basic Idea and Related work 41
5.2.1 NIDS and Multiple pattern matching 41
5.3 Proposed Design 43
5.3.1 Bloom Filter basics and overview 44
5.3.2 Scenario 1: With fixed sized matching patterns: 45
5.3.3 Scenario 2: With variable sized matching patterns: 46
5.4 Implementation 47
5.4.1 Implementation Constraints 48
5.4.2 Partial and Large Bloom Filter 48
5.4.3 Hash Function 50
5.4.4 Results and Comparison 51
5.5 Conclusion 53

6 Conclusion and Future Work 56

Bibliography 57

Dissemination of Work 63

List of Figures

2.1 Data flow of IDEA Cipher 9
2.2 IDEA Encryption key Generation 11

3.1 Architecture of six-stage pipelined new modulo $(2^{16}+1)$ multiplier for IDEA 27
3.2 Multiplier giving output in the 7th clock cycle 28

4.1 A single inner round pipelined architecture for IDEA with 24 pipeline stages 32
4.2 Basic Iterative architecture with inner round pipelining for IDEA . 33
4.3 Partial mixed inner and outer round pipelined architecture for IDEA 33
4.4 Full mixed inner and outer round pipelined architecture for IDEA . 35

5.1 Basic Architecture for Signature based NIDS 41
5.2 A Typical Bloom Filter with an Analyzer 45
5.3 Parallel Bloom Filter Matching a fixed sized pattern [1]. 46
5.4 A series of Bloom Filters matching variable sized patterns at a time [1]. 47
5.5 Partial Bloom Filter accepting 2 Hash Functions [2] 49
5.6 Large Bloom Filter using a series of PBFs [2] 50
5.7 Test-bench for the Hash Function Generator 51
5.8 Timing Summary for Hash Generator Module 52
5.9 Waveform for the overall design for Bloom Filter. Diagram shows the waveform when the supplied Hash values exactly matches with the hash values of the member string, the match signal becomes high 52
5.10 Timing Summary for the Bloom Filter 53

List of Tables

3.1 Device utilization and timing analysis for the proposed multiplier . 29

4.1 Comparison of the three different architectures implemented in FPGA. S is the number of total number of pipelined stages in the architecture, F is the clock frequency achieved by the design, T is the throughput of the design, N is the number of slices consumed by the design, R is the Throughput to Area ratio.'. 36

4.2 Comparison of our proposed design with some existing designs . . . 36

5.1 Comparison of our proposed design with some existing designs . . . 53

Chapter 1

Introduction

Motivation

Problem Statement

Our Contribution

Thesis Organization

Chapter 1

Introduction

In the field of networking, role of network security is immense. It is a very vital tool which provides the security against various external and internal threats in any network. To understand the theory of network security, the understanding and knowledge of different threats in the network. To maintain network security in a network involves the fulfilment of the security goals in the network which are **Data Confidentiality**, **Integrity** , **Authentication** and **Non-Repudiation**. Data confidentiality is achieved by means of Cryptography. The aim of cryptography is to secure information so that only the intended parties can read the data. Cryptosystems had been developed for centuries. As computer technologies are getting advanced, more and more cryptographic applications are used. They are mainly used to support other applications which are very much sensitive to data security such as smart cards and commercial data exchange over a network. Not only for personal use but cryptographic algorithms are also very important in every aspect of professional activities. A cryptographic algorithm generally consists of some specialized arithmetic computations which are complicated in terms of time complexity. It is because of the fact that these algorithms work with large amount of data either in blocks or simply in streams. Although a single traditional CPU is enough for performing these computations, but for a machine which works as a server in a huge network gets millions of client requests for performing cryptographic operations for them individually. This makes the workload huge. The computational resources may also be limited for example in smartcards, mobile phones, handheld computers, etc. Moreover if the associated network is of high

speed, the speed of the necessary cryptographic computations also needs to be taken into account. For example in transmitting audio and video data for cable TV, video conferencing and sensitive financial and commercial data, the speed of the cryptographic module to be embedded ,needs to be very high. Moreover for security related issues in wireless and sensor networks, there is a need for separate hardware device with very high processing rate because of limited battery of the nodes and for optimizing the bandwidth efficiency. So from the viewpoint of high speed and throughput, traditional software implementations of these complicated cryptographic algorithms are not efficient in real time applications like ATM, VPN, etc. This forces the system designers to go for hardware implementation of the cryptosystems.

1.1 Motivation

The demand for new network security systems is increasing with the growth of network services in our society. Several heuristics are associated for judging the network performances. In today's world, speed is considered as one of such heuristics. Moreover various network applications are being installed for reducing the network overhead, specially the data traffic in remote servers. Such types of applications are basically used for satisfying the security goals, namely confidentiality, integrity, authentication and non repudiation. Majority of these applications are very much computationally intensive and software approach for these applications is rather inefficient to work in line speed of the network. So the current trend is to replace such software applications with specialized hardware which are quite compatible to work in such high speed. There are certain key points which can be chosen as motivation for this work. These are as follows.

- For maintaining security over Internet and E-Commerce, cryptography and security is of vital importance.
- For performing cryptographic and other network security management operations, some heavy computations are necessary and performing them at a line speed of the network is a challenging task. Software based applications

cannot guarantee such speed, so there is an increasing demand for hardware based appliances.

- FPGA technology allow reconfigurability in any design and at the same time any architecture can exploit parallelism and concurrency when implemented in FPGA. So a lot of research work is now based on FPGA implementations of cryptographic algorithms.

- Finally one algorithm can be implemented in a variety of architectures, each having some speciality. By varying the architectures, different design objectives can be achieved which increases the flexibility of the algorithm.

1.2 Problem Statement

The main objectives of our work is to design, synthesize and verify the functionality of certain modules that are highly computationally intensive and are highly challenging to make them work as an independent module in a high speed network. This can be said in detail as :

1. In the context of confidentiality, the motive is to design an efficient architecture for a symmetric key block cipher and implement using FPGA technology. The target is to reduce the modular complexity of the round operations of the cipher so that the performance parameters can be optimized. This is to support the argument that building a cryptosystem completely on an FPGA platform is possible.

2. Exploit the concurrent characteristics of FPGA to include Outer Round and Inner Round Pipelining in our design. Finally the goal is to visualize their impact on system performance.

3. Design a novel architecture for a multiple pattern string matching algorithm which can be used as a pattern matching module in a Network Based Intrusion Detection System(NIDS).

1.3 Our Contribution

In this work, a set of computationally intensive modules are designed and and their performances are evaluated so as to verify their functionality as a separate application in a high speed network. For each and every design, the ultimate goal is to optimize the performance parameters i.e. to maximize the system throughput by maximizing the operating frequency of the design and to minimize the area requirements so that it validates the $area \times time^2$ [3] complexity .The modules include a symmetric key cryptographic co-processor where IDEA encryption algorithm is used and a Network Intrusion Detection System (NIDS) with a novel multiple pattern string matching technique. These modules are designed and implemented using efficient pipelined architectures for better performance. The main contributions of this thesis can be given as:

- In IDEA cipher, each round operation needs four modulo $(2^{16}+1)$ multipliers and efficient design of such multipliers is a challenging task. In our work, we have deigned a new pipelined architecture using higher radix Booth's algorithm which reduces the number of partial products as well as the intermediate operand sizes. For 16 bit operands, the proposed multiplier gives the output with a considerable low latency and thereby reduces the round complexity of the cipher.

- After reducing the round latency, the IDEA cipher is implemented and verified functionally with different combinations of outer and inner round pipelining so as to increase the system performance. The design achieves a fairly high throughput and it supports the statement that FPGAs are good choice for implementing cryptosystems on a single chip.

- A novel string matching architecture is designed using a memory efficient multi hashing data structure called Bloom Filter. The architecture is found to be suitable for a multiple pattern matching module in a NIDS. The speciality of this string matching module is that, there is no possibility of presence of false negatives i.e. a malicious packet cannot escape as a genuine packet

from the module.However, there may be possibility of false positives in the system.

1.4 Thesis Organization

In this chapter, the motivation for hardware implementation of networks security applications, the objectives of our work and our contribution is discussed in a nutshell. The organization of the rest of the thesis and a brief outline of the chapters in this thesis are as given below.

In **chapter 2**, we have discussed the basic algorithm and theory of IDEA, the symmetric key block cipher, which is chosen for implementation in FPGA. Moreover we have discussed the previous implementation details of IDEA in hardware.

In **chapter 3**, we have described our proposed modulo multiplier architecture using Diminished-one number representation and Booth's algorithm and analyzed how it reduces the modular complexity in IDEA round operations.

In **chapter 4**, we have described the performance of IDEA on FPGA using our proposed multiplier and a combination of outer and inner round pipelining. We analyzed our architecture in terms of Alice counts and throughput.

In **chapter 5**, we have proposed and implemented a novel string matching algorithm for a Network Based Intrusion Detection System using a memory efficient multi hashing data structure called Bloom Filter.

Finally, in **chapter 6**, we draw our conclusion and proposed some additional ideas for our future work.

Chapter 2

Background

Introduction

Symmetric key Cryptosystem

Previous implementations of IDEA in Hardware

Conclusion

Chapter 2

Background

2.1 Introduction

In this chapter, we have discussed the basic theory and algorithm for International data Encryption algorithm (IDEA), a symmetric block cipher, which is selected as a cipher for implementation in FPGA technology. This chapter consists of two parts. In the first part, the IDEA algorithm and basics have been discussed. In the second part, a detailed literature study of the previous implementations of IDEA in hardware is covered.

2.2 Symmetric key Cryptosystem

In symmetric key cryptography, the encryption and decryption process is done by the same key which is the secret key. This secret key is shared prior to the encryption process and remains constant during the process. In our work, we have used International Data Encryption Algorithm as the symmetric key block cipher module and we focussed mainly on reducing the round complexity and increasing the throughput for the overall design.

2.2.1 IDEA Encryption Algorithm

The proposed Encryption Standard (PES) is a block cipher introduced by Lai and Massey [4, 5]. It was then improved by the Lai, Massey and Murphy in 1991. This version, with stronger security against differential analysis and truncated differentials, was called the Improved PES (IPES). IPES was renamed to be the

International Data Encryption Algorithm (IDEA) [6] in 1992. Claims have been made that the algorithm is the most secure block encryption algorithm in the public domain.

Basic Structure

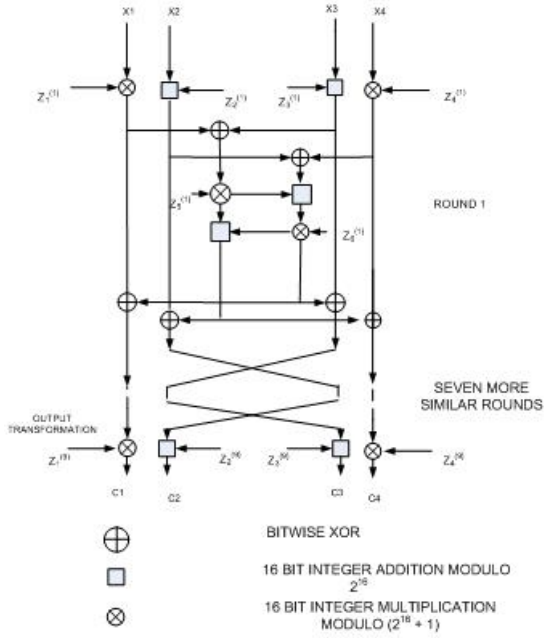

Figure 2.1: Data flow of IDEA Cipher

IDEA is a symmetric, secret-key block cipher. The keys for both encryption and decryption must be kept secret from unauthorized persons. Since the two keys are symmetric, one can divide the decryption key from the encryption one or vice versa. The size of the key is fixed to be 128 bits and the size of the data block which can be handled in one encryption/decryption process is fixed to 64 bits. All data operations in the IDEA cipher are in 16-bit unsigned integers. When processing data which is not an integer multiple of 64-bit block, padding is required. The security of IDEA algorithm is based on the mixing of three different kinds of algebraic operations: EX-OR, addition and modular multiplication. IDEA is based upon a basic function, which is iterated eight times. The first iteration operates on the input 64-bit plain text block and the successive iterations operate on the 64-bit block from the previous iteration. After the last iteration, a final

transform step produces the 64-bit cipher block.The data flow graph is shown in Figure 2.1. The algorithm structure has been chosen such that, with the exception that different key sub-blocks are used, the encryption process is identical to the decryption process. IDEA uses both confusion and diffusion to encrypt the data. Three algebraic groups, EX-OR, addition modulo 2^{16}, and multiplication modulo $(2^{16}+1)$, are mixed, and they are all easily implemented in both hardware and software. All these operations operate on 16-bit sub-blocks.

Key Generation

The key generation phase of IDEA generates 52 sub-keys from the 128 bit input key. The block diagram for key generation is shown in Figure 2.2. The basic steps of generating the encryption keys are:

- All the sub-keys are named as $Z_1^{(1)}$,...,$Z_6^{(1)}$,$Z_1^{(2)}$,...,$Z_6^{(2)}$,....,$Z_1^{(8)}$,...,$Z_6^{(8)}$, $Z_1^{(9)}$,...,$Z_4^{(9)}$.
- From the input 128 bit key, eight sub-blocks of 16 bits are partitioned and are assigned to $Z_1^{(1)}$,...,$Z_2^{(2)}$ directly.
- Now the original 128 bit key block is rotated by 25 bits and a new 128 bit block is formed. Now another eight sub-blocks are generated from this new block.
- The rotation procedure is repeated until and unless sub-blocks used in previous rounds are found.

Once the encryption keys are generated, the decryption keys can be generated directly by taking their additive inverse modulo 2^{16} and multiplicative inverse modulo $(2^{16}+1)$ as required.

2.3 Related Work

In high speed applications, where there is a need of protection of data, cryptographic algorithms are necessary. Data rates in such applications are very high and such computation cryptographic algorithms need to be run on real time so

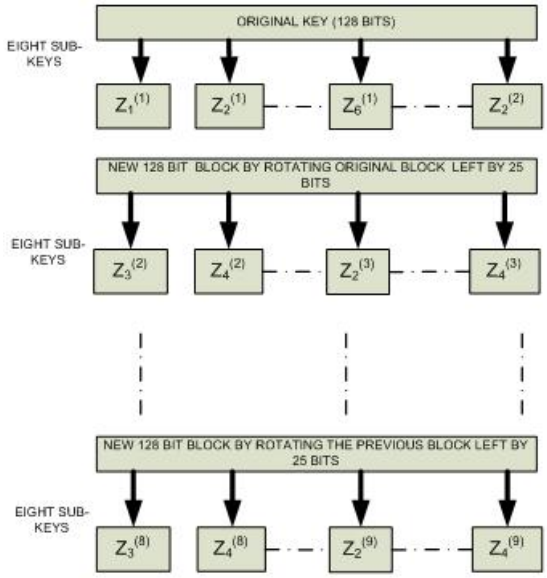

Figure 2.2: IDEA Encryption key Generation

as to provide the quality of service.In this scenario, a software implementations of such algorithm using general purpose processors due to delay in instruction processing.But such speed can be easily achieved when implemented in hardware.Although the software implementation is less costly than hardware implementation, the speed up in hardware is very high.So for flexibility,availability and high functionality, there is a need of incorporating a separate cryptographic module in such applications.

Although IDEA involves only simple 16-bit operations, software implementations of this algorithm still cannot offer the encryption rate required for online encryption in high-speed networks.IDEA has been previously implemented in hardware using various FPGA devices and even ASIC. Like other renowned symmetric key block ciphers, IDEA contains no S-Boxes or P- Boxes. So there is a less memory overhead.Instead it has some basic building blocks like EX-OR, addition modulo 2^n and multiplication modulo $2^n + 1$. Among these basic operations, the EX-OR and the addition modulo 2^n implementations are very straightforward.The multiplication module is the most computational intensive module and it needs a lot of effort to design it efficiently. In each round of IDEA, four such modulo multipliers are needed.So the performance of IDEA in hardware i.e. the through-

put rate and the area and cost efficiency depends a lot on efficient design of the multiplier.

IDEA was first implemented and verified in VLSI by Bonnenberg [7] where the data encryption and decryption was performed on a single hardware unit which was a 1.5 μm double metal n-well CMOS with a maximum clock frequency of 33 MHz and data throughput rate of 44 Mb/s.In this implementation, the key management module and the inversion module was not performed on chip.The main goal was to achieve the highest possible throughput along with a hardware support to verify whether the design was cryptographically correct in terms of functionality and availability. By that time , some effective architectures for modulo (2^n+1) multipliers were proposed by Bonnenberg and Curiger [8].Among those architectures, Bonnenberg's scheme [7] used the $(n+1) \times (n+1)$ multiplication scheme with a pipeline of two stage. With a computation speed of 60 ns per multiplication, a two multiplier round architecture using pipelining for IDEA was used. The speciality of this approach is that, the architecture is made of one encryption/decryption unit and an input/output interface unit with each unit containing a RAM. Proper clock was used to match their speeds.The drawback of this design is that extra overhead is associated due to huge data transfer from on chip RAMs as well as regulating off chip traffic.Moreover, the design was not supportive for all standardized modes of the cipher.Although it is the first VLSI implementation of IDEA, the data throughput rate was found to be twice than that of a DES chip at that time.

Bonnenberg's design [7] was found to be a prototype for a VLSI circuit , which was made essentially to speed up the cryptographical tests.But there was still a demand for a real time application hardware that can handle data traffic in high speed networks.The goal was to design an efficient basic building block with a high throughput datapath architecture with an efficient interface that can handle off-chip data traffic.

Curiger's implementation [9] [10] of IDEA was done on double-metal CMOS 1.2 μm which was suitable for all standardized modes.One of the speciality of this

implementation is that, the data encryption and decryption was implemented on a single hardware unit.With a system clock frequency of 25 MHz the data throughput rate was found to be 177 Mb/s.This was the first silicon block which was found compatible for online encryption in high speed networks.The design was made using eight pipelining stages, containing a single round to achieve temporal parallelism.

As usual, the design of modulo $(2^n + 1)$ multiplier was crucial for the performance of the cipher. Various multiplication schemes were defined in [8].Curiger's design used the multiplication scheme with modulo $(2^n + 1)$ adders in which one of the operands (say X) was in diminished-1 representation proposed in [11] and another operand was in normal weighted form which can be given as:

$$Z = \chi Y \ mod(2^n + 1) = (\sum_{i=0}^{n-1} 2^i \chi_i \ . \ \sum_{i=0}^{n} 2^i y_i) \ mod(2^n + 1) \tag{2.1}$$

$$= (\sum_{i=0}^{n} y_i (2^i \chi \ mod \ 2^n \ + \ 2^i \overline{\chi} \ div \ 2^n \ + \ 1) \ mod(2^n + 1)) \ mod(2^n + 1)$$

where

$$\overline{\chi} = \sum_{i=0}^{n-1} 2^i \overline{\chi_i}$$

and χ is a diminished-1 representation of $X = \sum_{i=0}^{n} 2^i x_i$, i.e. $\chi = X - 1$.

Later, in [10] a new approach was taken to avoid high computation time and area.A modified Booth recoding multiplication and a fast carry select additions for the final modulo correction were used as two stages of the multiplier in a pipeline structure. Four such modulo multipliers were used in each round for optimizing the performance of computational units.Each of the multiplication units used two stages of the eight stage pipeline. The design was made on a single hardware chip where the sub-keys were generated internally along with necessary computation of additive and multiplicative inverses.The multiplicative inverses were calculated using square and multiply method.Only the master key was loaded onto the chip at the beginning. So the speciality of this design was that, no off chip data traffic

was needed to manage through buffers.The overall architecture of Curiger's design [10] contains two on chip buffer memory for implementing the different modes of operation of the cipher.In each buffer, a 8×64 bit shift register is used for implementing eight stage pipeline.

In VLSI circuits, arrival of temporary or permanent faults are very common which creates error in encryption. To get rid of these faults, necessary fault detection tests are required. These tests can be off-line or online. But if these tests are periodic, it consumes unnecessary clock cycles and degrades the speed. For testing the overall functionality during encryption,an online built in self test scheme was added which was done by incorporating a fifth multiplier in the pipeline circuitry.The drawback in this approach was that, this hardware redundancy resulted in a large extra hardware. Moreover if the time between two tests was long, there was a probability of some short-lived error to creep in. Although, the proposed design was not the fastest single chip implementation but it was the first design which was found compatible for use in high speed networks.

Although the design proposed in [10] was compatible for real time encryption in high speed networks, there was still a demand for hardware with faster encryption ability. Moreover Curiger's design [10] was not capable of detecting all possible errors during its normal operation.Wolter's design [12] of a new hardware for IDEA was motivated by the requirement of higher data rate and online testing of circuit.The design was done by implementing one round of IDEA in a 0.8 μm CMOS and a data throughput of 355 Mb/s was obtained.The characteristics of the architecture was that, all the standardized modes of the cipher were capable to processing data with equal speed. The design of modulo $(2^{16}+1)$ multiplier was based on low high theorem of Lai and Massey [4] where modified Booth encoded multiplication algorithm and wallace tree were used. Here a 10 stage pipelining was used where two stages were reserved for performing online test.

For detecting faults, both off-line and online built in self test schemes were used in this scheme.The off-line test was performed using pseudo-random data encryption.Two online tests were performed, one based on information redundancy

i.e mod 3 residue code and the second test was based on redundant test words. Although this scheme has a data throughput rate of 355 Mb/s by implementing a single round, due to overhead of online tests on chip, some additional area requirements was required in this design.

The next implementation of IDEA was made by Salomao [13] on an Application Specific IC named HiPCrypto using a 0.7 μm two metal, which was oriented towards computer network applications (like VPN) demanding high throughput.A single chip was used and the operation frequency was 53 MHz clock.This design was made to meet the requirement of applications in current and future high speed data networks.For this, temporal and spatial parallelism was exploited on the main design. No built in self test schemes were incorporated in this design for detecting faults. The modulo $(2^n + 1)$ multiplication was designed using a two stage pipelined multiply unit. Four small RAMs were used for storing the sub-keys. By using a single HiPCrypto device, a data throughput rate of 424 MB/s was achieved by the design. The disadvantage of this scheme was that, the HiPCrypto chip was not able to handle sub-keys derived from multiple keys.

A paper design of IDEA processor using four xilinx XC4020XL devices was proposed by Mencer [14] and that proposed design achieved a data throughput rate of 528 Mb/s. The design was done for comparing the parameters like performance, programmability and power for ASICs, FPGAs and normal processors. During the FPGA implementation, 56 stage pipelining was exploited for performance improvement and a custom designed Konstant coefficient multiplier was used which was based on look-up tables.The limitations of this design was the prior loading of keys before encryption.

Leong [15] implemented the IDEA cipher using a bit serial architecture [16].Due to the bit serial architecture, the algorithm of the cipher was deeply pipelined. The operation frequency of this design was 125 MHz and a Xilinx Virtex XCV300-6 device was used. The data throughput rate was found to be 500 Mb/s which was as usual compatible for online encryption for high speed networks.The advantages of this implementation were :

- High degree of fine grain parallelism.
- Scalable and thus the trade-offs between data conversion rate and the area can be addressed.
- High clock rate.
- Compact implementation.

The design for the modulo $(2^{16} + 1)$ was done using the approach proposed by Meier and Zimmerman and described by Curiger [9] in which, modulo 2^n adders were used along with bit-pair recoding algorithm. To increase the throughput, a 16 stage pipelined version of Lyon's serial parallel multiplier was used. The overall design of the cipher was done using four parallel to serial converters and four four serial ro parallel converters. The key storage and subkey generations in each round was done using shift registers. The proposed design was found to be scalable using more resources.

For incorporating efficiency in reconfigurable computing, Goldstein [17] implemented the IDEA cipher on Piperench architecture and achieved a data throughput of 1013 Mb/s. Although, the design was more suitable for stream based applications,the speciality of the Piperench architecture was the improvement of compilation and reconfiguration time from normal FPGAs by means of an advanced computing technique called pipeline reconfiguration.This feature is one type of a hardware virtualization in which, the compiler is free from hardware constraints.The simulation of [17] was done by dataflow intermediate language.

Ascom, the patent holder of IDEA, implemented a commercial design of IDEA cipher called IDEACrypt kernel on 0.25 μm CMOS technology and achieved a throughput rate of 720 Mb/s.

Mosanya [18] implemented IDEACore, an encryption core for International Data Encryption Algorithm as a modular and reconfigurable cryptographic co-processor. The goal of that design was to accelerate cryptographic operations on a host system.The system was implemented using VHDL and it exploited the property of partial reconfiguration for a normal FPGA.In the multiplication mod-

ule, bit parallel multiplier was used and for modulo $(2^{16}+1)$ correction, low-high algorithm [4] was used. For the overall design, a scalable pipeline was used where the number of pipeline stages were decided during compilation time. The design achieved a throughput rate of 1500 Mb/s.The drawback of the scheme is that, the area requirements is not fixed every time due to variation of pipelined stages.Moreover, due to session initialization and key calculation, the overall performance is slightly low in this scheme.

Cheung and Leong [19] further implemented IDEA on a bit parallel architecture [16] on Xilinx Virtex XCV300-6 FPGA and achieved a data throughput of 1166 Mb/s at a system clock rate of 82 MHz.The implementation was runtime reconfigurable and by direct modification of bitstream downloaded to the FPGA, the key scheduling was done.Moreover, the implementation was scalable with increased resource requirements.With a full hardware support, a throughput of 5.25 Gb/s was estimated using this design.

With a fully pipelined approach, IDEA was implemented by Hamalainen [20] using Xilinx XCV1000E-6BG560 FPGA and the throughput of 8 Gb/s throughput was achieved by the design.The modulo multiplier used diminished-1 number representation [11] and the multiplication schemes used in [21] [22], [8], [23] were implemented and compared.Finally, the multiplication scheme of [22] was chosen. For cyclic left shifts, extra combinational logic was used and Carry save adders were used for multi-operand addition.The entire design was made using loop unrolling architecture but it was slightly less efficient in terms of area requirements.

Till now, the fastest FPGA implementation was done by Gonzalez [24] where a throughput of 8.3 Gb/s was achieved using Xilinx Virtex XCV600-6 device. The speciality of this design was that, all the operational units were replaced by constants and a partial reconfiguration was used along with superpipelining.The only drawback for this scheme is that, not many devices support partial reconfiguration.

Using embedded multipliers, IDEA was implemented by Pan and a throughput of 6 Gb/s was achieved but the design was costly in terms of area efficiency.An

efficient VLSI implementation of IDEA was done by Thaduri [25] using Altera FPGA, where the modulo multiplier was optimized by using Wallace trees and carry look ahead adders and a deep temporal parallelism was exploited.The speciality of the design is that, the sub-keys are generated internally once the original key is fetched.Moreover, the design did not use any additional RAM for storing the subkeys.Using a clock frequency of 10 MHz, a throughput greater than 700 Mb/s was achieved by the design. In terms of scalability, a throughput of 7.8 Gb/s was achieved using scaling.

2.4 Conclusion

In this chapter, the architecture and algorithm for IDEA cipher is discussed in details. Moreover a background study on the previous hardware implementations of IDEA has been discussed in details and an analysis is drawn in terms of modular complexity. In the next chapter, we will discuss our proposed multiplier and its architecture as well as complexity.

Chapter 3

Modulo multiplier for IDEA Cipher

Introduction

Diminished-one Number Representation

Algorithm for the proposed multiplier

Proposed multiplier architecture

Complexity of the proposed multiplier

Conclusion

Chapter 3

Modulo $(2^{16}+1)$ multiplier for IDEA Cipher

3.1 Introduction

The basic design goals for hardware implementation of any algorithm is to reduce the time complexity for fast execution, optimization of basic modules and reducing the response time of the algorithm. IDEA is based on three algebraic group operations on 16 bit unsigned integers which are EX-OR, addition modulo 2^{16} and multiplication modulo $(2^{16}+1)$ [26]. Among these, the multiplication module is the most complex module because of 16 bit multiplication and modulo correction. Thus efficient design of these multipliers is a major issue for optimizing the performance of IDEA. In our design, we have proposed a new architecture for the multiplication module which is based on diminished-one number representation and radix 8 Booth's recoding algorithm. This multiplier generates less number of partial products as compared to previous implemented multipliers and there is no extra overhead for modulo correction.

3.2 Diminished-one Number Representation

The diminished-one number representation proposed by Leibowitz [27], is a very convenient and efficient form of representation of binary numbers in arithmetic modulo (2^n+1).In IDEA, all intermediate operands are 16 bit unsigned integers but for implementing modulo $(2^{16}+1)$ arithmetic in hardware, the register size

needs to be (16+1) bit. So unnecessarily an extra bit is used. To avoid this inconvenience, normal binary operands are transformed to diminished-one operands by subtracting one from normal binary representation of any number. So if A is an n+1 bit binary number, then the diminished-one representation of A which is an n bit number and denoted by d[A], is given by

$$d[A] = (A-1)\ mod(2^n+1) \tag{3.1}$$

Thus if $A \in [1, 2^n]$ and $A \neq 0$, then $d[A] \in [0, 2^n - 1]$,which is an n bit number.However when $A = 0$, $d[A] = d[0] = (0-1)\ mod\ (2^n+1) = (-1)\ mod\ (2^n+1)$ which is equal to 2^n, an $(n+1)$ bit number.

3.2.1 Basic Operations

The diminished-1 represented numbers follow some basic operations which are defined below:

$$d[-A] = \overline{d[A]} \quad if\ d[A] \in [0, 2^n - 1] \tag{3.2}$$

$$d[A+B] = (d[A] + d[B] + 1)\ mod\ (2^n+1) \tag{3.3}$$

$$d[A-B] = (d[A] + \overline{d[B]} + 1)\ mod\ (2^n+1) \tag{3.4}$$

$$d[AB] = (d[A] \times d[B] + d[A] + d[B])\ mod\ (2^n+1) = (d[A] \times B + B - 1)\ mod\ (2^n+1) \tag{3.5}$$

$$d[2^k A] = iCLS(d[A], k) \quad if\ d[A] \in [0, 2^n - 1] \tag{3.6}$$

$$d[-2^k A] = iCLS(\overline{d[A]}, k) \quad if\ d[A] \in [0, 2^n - 1] \tag{3.7}$$

where $\overline{d[A]}$ is one's complement of d[A] and iCLS(x,k) is the k bit circular shift of x in which the circulated k bits are complemented.For example if A is $(a_{n-1}a_{n-2}a_{n-3}...a_2a_1a_0)$ then $d[2^3A]$ is $(a_{n-4}a_{n-5}...a_2a_1a_0\ \overline{a_{n-1}}\ \overline{a_{n-2}}\ \overline{a_{n-3}})$

3.3 Algorithm for the proposed multiplier

Previously, many modulo multipliers were proposed which used diminished-one operands. But those approaches did not consider the handling of zero inputs and giving the results.Although some of them used array multipliers,but the hardware

complexities was more. Chen and Yao [28] proposed modulo $(2^n + 1)$ multipliers for the diminished-1 representations where radix-4 booth's recoding was used as the multiplication algorithm. The number os partial products was reduced to n/2 and the zero correction module was also simple. Our proposed multiplier follows Chen and Yao [28] scheme and based on radix-8 Booth recoding and the number of partial products generated in less than n/2 for n bit multiplication, thereby reducing the number of intermediate addition operations.The correction term generator module is also modified in this scheme. For adding the partial products, an inverted End Around Carry (EAC) adder tree is used. Finally,one diminished-1 adder is used for generating the product. Let A and B be two n+1 bit binary numbers and let d[A] and d[B] be their respective n bit diminished-one representations,such that d[A]= $(a_{n-1}a_{n-2}a_{n-3}...a_2a_1a_0)$ and d[B] = $(b_{n-1}b_{n-2}b_{n-3}...b_2b_1b_0)$. So we have,

$$d[B] = (\sum_{i=0}^{n-1} b_i 2^i)\ mod(2^n + 1) \tag{3.8}$$

Taking radix value as 8, the above equation can be written as,

$$d[B] = \left|(b_0 + 2b1 - 4b_2) + \sum_{i=1}^{\lfloor n/3 \rfloor} (b_{3i-1} + b_{3i} + 2b_{3i+1} - 4b_{3i+2})2^{3i}\right|_{(2^n+1)} \tag{3.9}$$

So,we can write,

$$B = \left|(1 + b_0 + 2b1 - 4b_2) + \sum_{i=1}^{\lfloor n/3 \rfloor} (b_{3i-1} + b_{3i} + 2b_{3i+1} - 4b_{3i+2})2^{3i}\right|_{(2^n+1)} \tag{3.10}$$

Substituting the value of B in equation 5 we have,

$$d[AB] = \Bigg|d[A] \times (1 + b_0 + 2b_1 - 4b_2) + d[A] \times \sum_{i=1}^{\lfloor n/3 \rfloor} (b_{3i-1} + b_{3i} + 2b_{3i+1} - 4b_{3i+2})2^{3i}$$
$$+ \sum_{i=1}^{\lfloor n/3 \rfloor} (b_{3i-1} + b_{3i} + 2b_{3i+1} - 4b_{3i+2})2^{3i} + (1 + b_0 + 2b_1 - 4b_2) - 1\Bigg|_{(2^n+1)} \tag{3.11}$$

or,

$$d[AB] = \Bigg| \sum_{i=1}^{\lfloor n/3 \rfloor} (d[A] \times (b_{3i-1} + b_{3i} + 2b_{3i+1} - 4b_{3i+2})2^{3i} + (b_{3i-1} + b_{3i} + 2b_{3i+1} - 4b_{3i+2})2^{3i})$$

$$+d[A(1+b_0+2b_1-4b_2)]\Big|_{(2^n+1)}$$

or,

$$d[AB]=\Big|d[A(1+b_0+2b_1-4b_2)]+\sum_{i=1}^{\lfloor n/3\rfloor}(d[A(b_{3i-1}+b_{3i}+2b_{3i+1}-4b_{3i+2})2^{3i}]+1)\Big|_{(2^n+1)}$$

or,

$$d[AB]=\Big|d[A(1+b_0+2b_1-4b_2)]+\sum_{i=1}^{\lfloor n/3\rfloor}d[A(b_{3i-1}+b_{3i}+2b_{3i+1}-4b_{3i+2})2^{3i}]+\lfloor n/3\rfloor\Big|_{(2^n+1)} \tag{3.12}$$

Now if the value of n is divisible by 3, then equation (11) can be expressed as,

$$d[AB]=\Big|d[A(b_{n-1}+b_n+2b_{n+1}-4b_{n+2})2^n]+d[A(1+b_0+2b_1-4b_2)]+\frac{n}{3}$$
$$+\sum_{i=1}^{\frac{n}{3}-1}d[A(b_{3i-1}+b_{3i}+2b_{3i+1}-4b_{3i+2})2^{3i}]\Big|_{(2^n+1)}$$

or,

$$d[AB]=\Big|d[-A(b_{n-1}+b_n+2b_{n+1}-4b_{n+2})]+d[A(1+b_0+2b_1-4b_2)]+1+\frac{n}{3}-1$$
$$+\sum_{i=1}^{\frac{n}{3}-1}d[A(b_{3i-1}+b_{3i}+2b_{3i+1}-4b_{3i+2})2^{3i}]\Big|_{(2^n+1)}$$

or,

$$d[AB]=\Big|d[-A(b_{n-1}+b_n+2b_{n+1}-4b_{n+2})+A(1+b_0+2b_1-4b_2)]+\frac{n}{3}-1$$
$$+\sum_{i=1}^{\frac{n}{3}-1}d[A(b_{3i-1}+b_{3i}+2b_{3i+1}-4b_{3i+2})2^{3i}]\Big|_{(2^n+1)}$$

Considering $b_i=0, for\ i\geq n$, we have,

$$d[AB]=\Big|d[A(1+b_0+2b_1-4b_2-b_{n-1})]+\frac{n}{3}-1+\sum_{i=1}^{\frac{n}{3}-1}d[A(b_{3i-1}+b_{3i}+2b_{3i+1}-4b_{3i+2})2^{3i}]\Big|_{(2^n+1)}$$

or,

$$d[AB]=\Big|d[A(\bar{b}_{n-1}+b_0+2b_1-4b_2)]+\sum_{i=1}^{\frac{n}{3}-1}d[A(b_{3i-1}+b_{3i}+2b_{3i+1}-4b_{3i+2})2^{3i}]+\frac{n}{3}-1\Big|_{(2^n+1)} \tag{3.13}$$

Here $\bar{b}_{n-1}$ denotes the one's complement of b_{n-1}.The terms $(\bar{b}_{n-1}+b_0+2b_1-4b_2)$ and $(b_{3i-1}+b_{3i}+2b_{3i+1}-4b_{3i+2})$ used in equation (12) are based on Radix 8 Booth's recoding algorithm and can have the any one of the following values $\{-4,-3,-2,-1,0,+1,+2,+3,+4\}$. So the possible values of the terms $d[A(\bar{b}_{n-1}+b_0+2b_1-4b_2)]$ and $d[A(b_{3i-1}+b_{3i}+2b_{3i+1}-4b_{3i+2})2^{3i}]$ can be $d[\pm A.2^{3i}]$,$d[\pm A.2^{3i+1}]$, $d[\pm A.3.2^{3i}]$, $d[\pm A2^{3i+2}]$ and $d[0]$.The values for $d[\pm A.2^{3i}]$,$d[\pm A.2^{3i+1}]$ and $d[\pm A2^{3i+2}]$ can be obtained easily from equation (6) and (7). The value for $d[\pm A.3.2^{3i}]$ is obtained as follows :

$$d[\pm A.3.2^{3i}] = d[\pm(A.2.2^{3i}+A.1.2^{3i})] = \left| d[\pm A.2^{3i+1}] + d[\pm A.2^{3i}] + 1 \right|_{(2^n+1)}$$

For this reason, an additional diminished-one adder is used as a separate module which generates the final partial product in this case.

To avoid the d[0] value,a correction term module has been proposed in the scheme which is a modification correction term module used in Chen and Yao's scheme [28].Corresponding to each partial product generated, a correction bit is generated. When n is divisible by 3, n/3 partial products(PPD) and correction bits(c) are generated which are based on the following condition:

- When i=0, if $(\bar{b}_{n-1}+b_0+2b_1-4b_2) \neq 0$, then $PPD_0 = d[A(\bar{b}_{n-1}+b_0+2b_1-4b_2)]$ and $c_0 = 0$ else $PPD_0 = 0$ and $c_0 = 1$.

- When $0 < i < \frac{n}{3}$, if $(b_{3i-1}+b_{3i}+2b_{3i+1}-4b_{3i+2})2^{3i} \neq 0$, then $PPD_i = d[A(b_{3i-1}+b_{3i}+2b_{3i+1}-4b_{3i+2})2^{3i}]$ and $c_i = 0$, else, $PPD_i = 2^{3i}-1$ and $c_i = 2^{3i}$.

Following the same approach as used in Chen and Yao scheme [28], we can write in the context of our scheme as,

- When n is divisible by 3,

$$d[AB] = \left| \sum_{i=0}^{\frac{n}{3}-1} PPD_i - \sum_{i=0}^{\frac{n}{3}-1} c_i + \frac{n}{3} - 1 \right|_{(2^n+1)} \tag{3.14}$$

- When n is not divisible by 3, there are two possible solutions

$$d[AB] = \left| \sum_{i=0}^{(\frac{n+1}{3})-1} PPD_i - \sum_{i=0}^{(\frac{n+1}{3})-1} c_i + \frac{n+1}{3} - 1 \right|_{(2^n+1)} \tag{3.15}$$

or,

$$d[AB] = \left| \sum_{i=0}^{(\frac{n+2}{3})-1} PPD_i - \sum_{i=0}^{(\frac{n+2}{3})-1} c_i + \frac{n+2}{3} - 1 \right|_{(2^n+1)} \tag{3.16}$$

In IDEA, all operands are 16 bit binary numbers.So the value of n is fixed(i.e 16). In that case, equation(17) satisfies the condition for multiplication.Equation(17) can be further written as,

$$d[AB] = \left| \sum_{i=0}^{Z-1} PPD_i - C + Z - 1 \right|_{(2^n+1)} \tag{3.17}$$

where

$$Z = \left(\frac{n+2}{3} \right)$$

$$C = \sum_{i=0}^{(\frac{n+2}{3})-1} c_i$$

$$c_i \in \{0, 2^{3i}\}$$

which can be finally extended as,

$$d[AB] = \left| \sum_{i=0}^{Z-1} PPD_i + \bar{C} + 2 + Z - 1 \right|_{(2^n+1)}$$

$$d[AB] = \left| \sum_{i=0}^{Z-1} PPD_i + \bar{C} + d[1] + Z + 1 \right|_{(2^n+1)} \tag{3.18}$$

Thus this newly proposed multiplier reduces the number of partial products by less than n/2.Moreover it has all the functionality to handle zero inputs and outputs as well as it avoids (n+1) bit arithmetic circuits during computation. The only overhead is the extra diminished-one adder for calculating the partial products for the terms $d[\pm A.3.2^{3i}]$. But efficiency of this design outweigh this extra logic overhead.

In IDEA, all the operands are of 16 bit size and no operand is treated as 0. All zero operands are taken as 2^{16}. So in that case, diminished-one form of that 0

operand is the diminished-one form of 2^{16}. For example, if A is a 16 bit operand in IDEA cipher, then A = 0 implies that A = 2^{16}. So $d[A] = |A-1|_{2^{16}+1} = 2^{16}-1$, not 2^{16}.

3.4 Proposed multiplier architecture

The newly proposed modulo $(2^{16}+1)$ multiplier consists of a Partial Product Generator (PPDG),a correction term generator, an Inverted End Around Carry CSA tree and two diminished-one modulo $(2^{16}+1)$ adder for generating partial product and final addition. The Partial Product Generator (PPDG) consists of a Booth's Encoder(BE), a Booth's Selector (BS) and one diminished-one modulo $(2^{16}+1)$ adder. The BE and BS follows a 4 bus approach. The BE module checks the overlapping quadruplet and generates the corresponding code. BS module takes the code as input along with the multiplicand and produce the partial product. The diminished-one modulo $(2^{16}+1)$ adder is used in this module for handling the value of $d[\pm A.3.2^{3i}]$. The Inverted End Around Carry CSA tree takes Z+2 operands and reduces it to two vectors, the sum vector and the carry vector. The individual adders are made of Full adders (FA). The correction term generator checks each code generated and generates the corresponding correction term. The final diminished-one modulo $(2^{16}+1)$ adder is used for adding the final two sum and carry vector. The major goal of this implementation is to achieve high throughput, which motivates to minimize the delay of the computation intensive modules of the design. As the multiplication module is the most time consuming in IDEA data path, pipelining mechanism is incorporated inside the multiplier. The design is mainly based on systolic approach in which pipelined registers forward data to the next stage in every clock cycle. For obtaining high performance, a seven stage pipelining is used in the design. The pipelined architecture of the multiplier is shown in Figure 3.1.

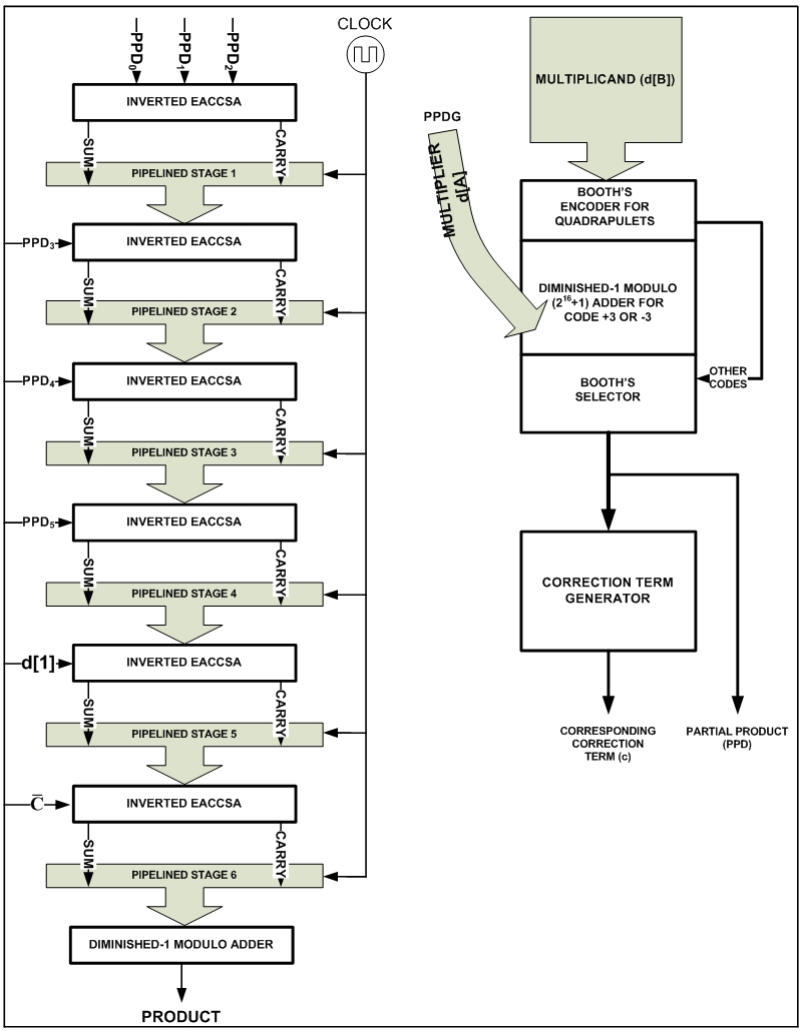

Figure 3.1: Architecture of six-stage pipelined new modulo $(2^{16}+1)$ multiplier for IDEA

3.5 Complexity of the proposed multiplier :

The qualitative comparison of the proposed multiplier is made using unit gate model as proposed by Tyagi [29]. According to this model, an Ex-OR/ Ex-NOR gate is charged 2 gate delay units and a delay through an elementary gate is taken as 1 gate delay units. The latency for the proposed multiplier consists of the delay of the PPDG module, the delay of the CSA tree and the delay of the final diminished-one adder. The PPDG module consists of BE, BS and one diminished-one adder. The delay in diminished-one adder as given in [28] is $2\lceil \log_2 n \rceil + 3$. So the final delay for the PPDG can be given as $T_{BE} + T_{BS} + T_{Dim-Adder}$. As $T_{BE} + T_{BS}$ has a constant delay (K) as per unit gate model, we have,

$$T_{PPG} = T_{BE} + T_{BS} + T_{Dim-Adder}$$

or

$$T_{PPG} = K + 2\lceil \log_2 n \rceil + 3$$

The CSA tree used in the design accepts $\frac{n+2}{3} + 2$ operands so the delay can be written as

$$T_{CSA} = T_{FA} \times H\left(\frac{n+2}{3} + 2\right)$$

Here H(x) is the height of the CSA tree using x number of inputs. The critical path delay for a full adder as per unit gate model is 4 units of time. From this, the overall delay of the multiplier can be written as

$$T_{Multiplier} = T_{PPG} + T_{CSA} + T_{Dim-Adder}$$

$$or, T_{Multiplier} = K + 2\lceil \log_2 n \rceil + 3 + 4 \times H\left(\frac{n+2}{3} + 2\right) + 2\lceil \log_2 n \rceil + 3$$

3.6 Results and Analysis

In this section, the waveform, device utilization report and the timing summary is discussed in brief.

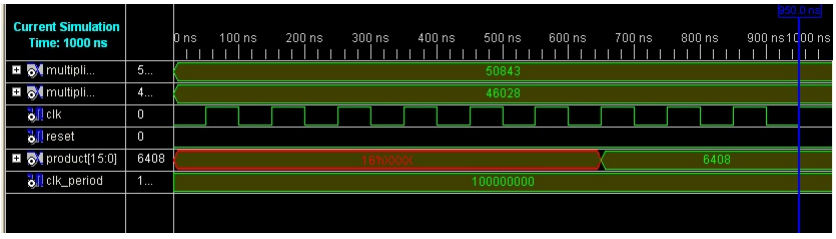

Figure 3.2: Multiplier giving output in the 7th clock cycle

In IDEA, all the operands inside a round are 16 bit unsigned integers. So for the multiplication module, both the multiplier and the multiplicand are 16 bit numbers. As shown in the multiplier architecture, for 16 bits, seven pipelined registers are used for getting the final product. So for getting the first output from the multiplier module, 7 clock cycles are consumed, each triggering a pipelined register sequentially. After getting the first output, in 7th clock cycle, the next

outputs are generated in the subsequent clock cycles i.e. eighth,ninth, etc. The waveform generated in the synthesis report for the multiplier module is shown in Figure 3.2. The device utilization summary and the timing analysis is given in Table 3.1.

Table 3.1: Device utilization and timing analysis for the proposed multiplier

Parameters	Values
Maximum Frequency	723.668 MHz
Device	Virtex 2 pro - XC2VP30
Number of Slices	496
Slices available	13696
Percentage of utilization	3

3.7 Conclusion

In this chapter, we have proposed and discussed a novel architecture for a modulo multiplier for the IDEA cipher. The multiplication approach is quite efficient in terms of number of partial products (which is less than $\frac{n}{2}$) and it uses Radix 8 Booth's multiplication algorithm. The design is made pipelined using 7 pipelined registers and the addition process is enhanced using inverted end around carry save adder tree. The synthesis report generated for the design shows that the design is quite efficient in terms of throughput and latency and can replace the internal multipliers of the target FPGA.

Chapter 4

Design and Implementation of IDEA

Introduction

Design and Architecture using pipelining

Result and comparison with other schemes

Conclusion

Chapter 4

Design and Implementation of IDEA cipher

4.1 Introduction

There are various goals for implementing any design in hardware as mentioned in [30]. For some designs, optimizing area requirements is a primary goals. For other designs where speed is an essential criteria, the objective is to increase the throughput and reduce the latency. The main parameters which are taken into account for implementing a block cipher in hardware are Encryption(Decryption) throughput and the circuit area. When large amounts of data are associated in any application, throughput is the best measure for the cipher speed. For applications with small data usage, latency is taken as an additional performance parameter, along with throughput. Circuit area usually determines the cost of implementation which helps to estimate the required $area \times time^2$ [3] balance for the design.

4.2 Design and Architecture using pipelining

For implementing any block cipher in hardware, the first step is to implement the basic iterative architecture first. Then new design strategies are added to the basic design like loop unrolling, outer round pipelining and inner round pipelining, as described in [30] so as to achieve the maximum required throughput.

Various new design methodologies of secret key clock ciphers have been discussed in [31].Our implementation follows three of these new design methodologies. All these designs are first synthesized in VHDL for functionality verification. Then they are implemented in Xilinx Virtex II Pro - XCVP30 FPGA. Primarily a single round is designed with optimum number of pipelined registers inserted inside the round. A similar type of design with inner round pipelining is followed in [32]. In

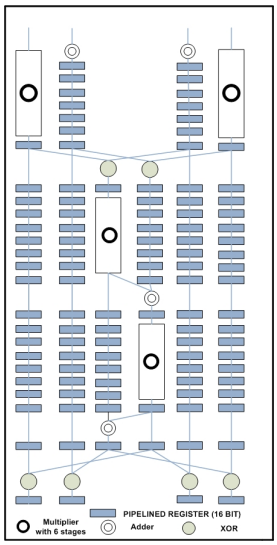

Figure 4.1: A single inner round pipelined architecture for IDEA with 24 pipeline stages

this design, a 24 stage pipelining is used which guarantees optimum throughput. However, as mentioned before, each multiplication module inside the round has 6 pipelined stages whereas for addition and XOR module, a single stage is used. The pipelined data flow in one round is shown in Figure.4.1. For this single round, the minimum clock period is found to be 9.749 ns in Virtex II Pro i.e the maximum clock frequency is 102.63 MHz. This design uses a slice count of 2340.

The first design implemented is the iterative design of this inner round pipelined single round. This is same like implementing a single round of IDEA and repeating it a number of times. The iteration is made for 8 rounds and a final output transformation round is implemented at last. The architecture of this design is given in Figure.4.2. The design is found to have a maximum clock frequency of 102.637 MHz with slice counts of 2340 in Virtex II pro . No outer round pipelined

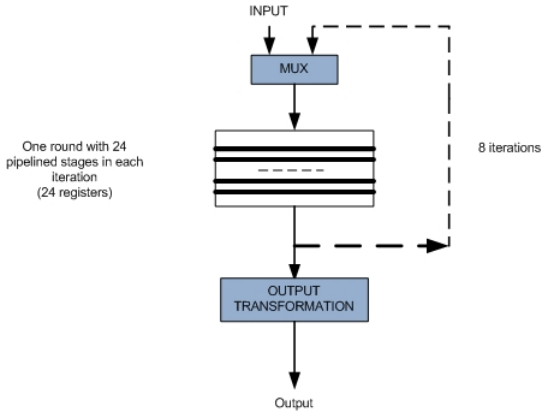

Figure 4.2: Basic Iterative architecture with inner round pipelining for IDEA

stages is used in this design. In each iteration, data flow through a single round with 24 stages of pipeline. The constraint of this design is that, new data for encryption can be fed into the system only after the completion of all the rounds.

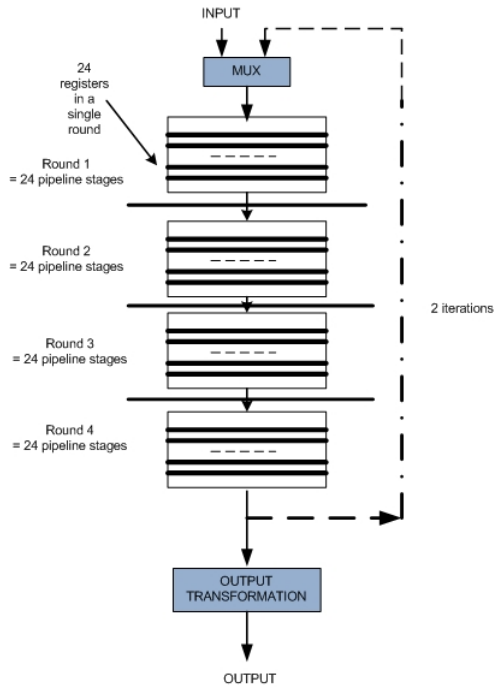

Figure 4.3: Partial mixed inner and outer round pipelined architecture for IDEA

In the second modified design, the design is based on partial mixed inner and outer round pipelining. This design has a much higher throughput with a marginal increase in circuit area. So in this case the throughput to area ratio

increases. In this design, 4 complete rounds are unrolled , each with 24 stages on inner pipelined stages. This 4 unrolled round design is iterated twice and the output transformation stage is added in the end. With this design, the maximum clock frequency is found to be 105.91 MHz with a consumption of 9471 slices in Virtex II pro. The architecture of this design is given in Figure.4.3 The number of pipelined stages used in this design is $[(24 \times 4) + 3] = 99$.

The third design is based on full mixed inner and outer round pipelining. This design is made so as to achieve the optimum throughput by increasing the clock frequency. Virtex II Pro doesn't have the required slices for all the IDEA rounds in unrolled basis. So only 6 rounds are unrolled along with the output transformation round. The design gives the throughput with a clock frequency of 117.61 MHz in Virtex II Pro. A total of $[(24 \times 6) + 5) = 149$ pipelined stages is used in this design. For placing all the rounds, a total of $[(24 \times 8) + 8) = 199$ pipelined stages As a result, it has more area requirement in terms of registers usage. The speciality of this design over the previous design is that, new data can be taken as input for encryption in every clock cycle, after that initial latency of the design. The architecture of the design is shown in Figure.4.4

4.3 Result and comparison with other schemes

In this section, needful results are analyzed and compared based on certain criteria. At first, the complexity of the proposed multiplier is defined. Next a detailed comparison is drawn for three different architectures implemented based on the performance parameters like throughput, slice counts and throughput to area ratio. Finally, the overall result is compared with some of the previous implemented schemes.

4.3.1 Analysis and Comparison

The comparison of the three architectures implemented are given in Table 4.1 for Virtex II pro FPGA device, which is made based on the performance parameters like slice counts, throughput and throughput to area ratio. The target device is

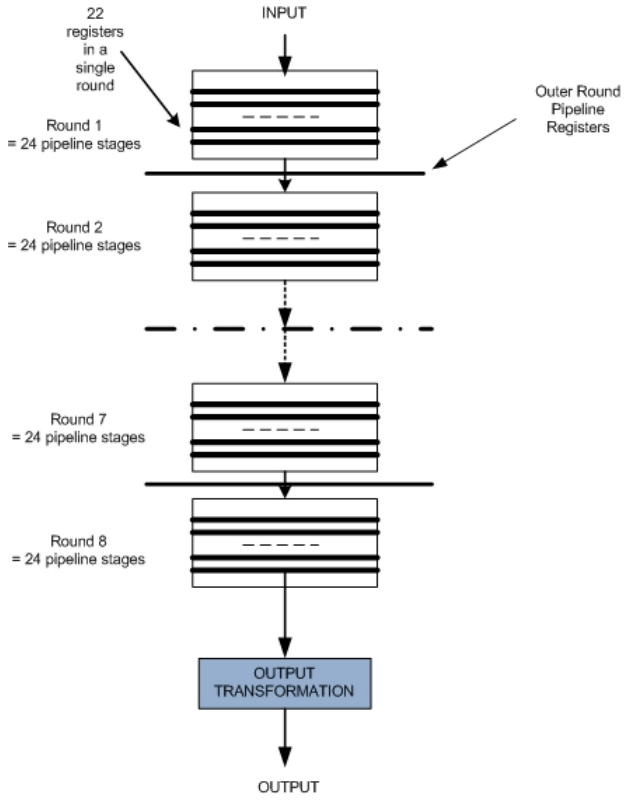

Figure 4.4: Full mixed inner and outer round pipelined architecture for IDEA

chosen as Virtex II pro . The design is synthesized using VHDL and the functionality verification is made using Chipscope pro. The comparison shows that as the number of pipelined stages increases, the throughput increases but the slice counts also increases to a certain amount. Due to this, the throughput to area ratio decreases slightly. However, this trade-off can be accepted as long as throughput is getting optimized. It is to be noted that in this design, no embedded multipliers are used and so there is no restriction in number of multipliers to be used. While implementing IDEA on Virtex II pro, using full mixed outer and inner round pipelining approach , it is found that all the rounds of IDEA cannot be accommodated in the device. So only 6 rounds are unrolled and implemented for verification.

A brief comparison of performance is made in Table 4.2 between our design and some existing designs of [33], [32] and [24]. The comparison is made based on the parameters like Throughput, Slice counts, Latency etc.

Table 4.1: Comparison of the three different architectures implemented in FPGA. S is the number of total number of pipelined stages in the architecture, F is the clock frequency achieved by the design, T is the throughput of the design, N is the number of slices consumed by the design, R is the Throughput to Area ratio.'.

Iterative		Partial Mixed		Full Mixed	
Virtex II pro		Virtex II pro		Virtex II pro	
S	24	S	99	S	199
F	102.637	F	125.91	F	153.8
T	821	T	3021.8	T	9843.2
N	2339	N	8117	N	11202
R	0.351	R	0.372	R	0.878

Table 4.2: Comparison of our proposed design with some existing designs

Design	Device	Throughput	Slices	Latency
Our design	(V2)XC2VP30	9.61 Gbps	11202	199-795 cycles
Granado's design	(V2)XC2V6000	27.948 Gbps	15016	182-833 cycles
Gonzalez's design	(V2)XCV600	8.3 Gbps	6078	158-1205 cycles
Hamalainen's design	(V2)XCV2000E	6.8 Gbps	8640	132-1250 cycles

4.4 Conclusion

In this chapter, we have proposed a new design for implementing IDEA cipher in hardware. We have incorporated a new design for modulo multiplication module which generates less than $\frac{n}{2}$ partial products and at the same time the partial products are generated by only circular lest shift of the multiplicand. Radix 8 Booth's recoding algorithm is chosen as the recoding algorithm for reducing the number of partial products and the operands are taken as diminished-one representation which helps to perform the modular correction implicitly. This multiplier is used in the IDEA algorithm and to increase the throughput, outer round and inner pipelining approach is used. The performance of the cipher using this design reveals that this design can be used for implementing IDEA on FPGA for High performance cryptosystem. As super-pipelined approach is used, this design can fully support Cipher Feedback(CFB) mode of operation with a high performance for the cipher. Moreover, by replicating the number of such FPGA devices, an overall design with a high throughput can be achieved. The

consumption of power has not been considered in this design at present. In future, the design may be extended to that with low power and high throughput.

Chapter 5

FPGA based string matching for NIDS

Introduction

Basic Idea and Related work

Proposed Design

Implementation

Conclusion

Chapter 5

FPGA based string matching for Network Intrusion Detection System

5.1 Introduction

The field of network security guarantees the prevention and monitoring of unauthorized access, misuse, modification as well as denial of network accessible resources. The major goals of network security includes confidentiality, data integrity, authentication and non repudiation. In the same context, intrusion detection is a security management tool which monitors network traffic for detecting possible security breaches. These security breaches attempt to compromise the confidentiality, integrity or availability of network resources and can be either from outside or inside the network concerned. In traditional networks, firewalls are used to monitor and filter incoming and outgoing packets but they cannot eliminate all security threats, nor they can detect attacks when they happen. It is like a locked gate to a treasure house that prevent the entry of thieves. Network Intrusion Detection System (NIDS) is another network processing application, which is either a software application (example Snort) or a hardware device that monitors network for malicious activities such as denial of service attacks, port scans etc. This NIDS along with Network Intrusion Prevention System (NIPS) are essential network security appliances that helps in maintaining the security goals in a network to a great extent. Intrusion Detection and Prevention Systems (IDPS)

are primarily focused on identifying possible incidents, logging information about them, attempting to stop them, and reporting them to security administrators. As the main work of such systems is to monitor network traffics for suspicious activities or patterns, they can be regarded as a multiple pattern matching or string matching module. String matching algorithm is thus one of the most critical module in such systems and the detection of intruders are performed based on this module. At the higher level, there are management softwares which configure, log and display alarms. A database for a number of malicious patterns is maintained at the back end.Whenever any packet containing such malicious patterns is found during packet monitoring, the detector engine of NIDS raises an alert call to the administrator for taking necessary action against the target packet.

Although the decisive factor for a NIDS is a multi pattern string matching algorithm, it is a highly challenging task to implement. It is because the operation of a typical NIDS involves deep packet inspection [34].Checking every byte of an incoming packet in a network to see if it is matching with one of a set of thousands of patterns becomes a computationally intensive task. Moreover if it is a high speed network, packet inspection needs to be performed in line speed which is more challenging. Software based approach is not efficient in terms of speed and moreover parallelism cannot be exploited in case of multiple pattern matching. So specialized hardware approaches are required to maintain match up with the network speed and to maintain a tight bound on worst case performance.

In this chapter, a new architecture for a multiple string matching based NIDS, is presented. The string matching module consists of a memory efficient multi hashing data structure called Bloom filters, which can detect strings in streaming data without degrading network throughput. Moreover new rules or patterns can be added without interrupting the normal operations. The speciality of our design is that, multiple patterns can be matched in a single clock cycles unlike traditional software based string matching algorithms. Furthermore the computation time involved in performing the query is independent of the number of patterns in the database. The following sections in this chapter describes the background

and the basic ideas of NIDS which is followed by our proposed string matching algorithm using Bloom filters. Two scenarios are further discussed on the context of the length of the input packet and the multiple length patterns. Finally the implementation details and analysis are discussed which is followed by the conclusion.

5.2 Basic Idea and Related work

In this section, the basic idea and architecture for a typical Network Intrusion Detection System is discussed followed by the previous implementations.

5.2.1 NIDS and Multiple pattern matching

As discussed before, a NIDS is simply a software application or a specialized hardware which monitors the network packets for malicious activities. It maintains a database of fixed or variable sized patterns which are searched against an input data packet. A basic architecture for a NIDS is given in Figure 5.1. At the top level, it works as an alarm in the network but at the core , it is computationally challenging as it requires deep packet inspection and that too in network speed. For a high speed network, deep packet inspection signifies that every byte of every packet must be searched for multiple patterns. So in a nutshell, the operation is nothing but a multiple pattern matching algorithm.

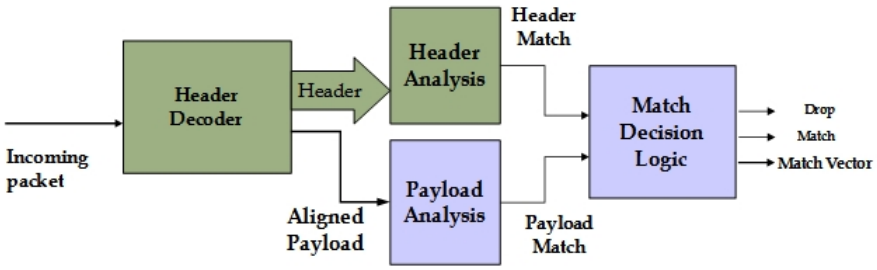

Figure 5.1: Basic Architecture for Signature based NIDS

Scanning or monitoring a packet in a network involves both header as well as payload analysis of the packet. The overhead for the header analysis is much less

than the payload analysis and the header size is fixed unlike the packet payload.

String Matching Engine: The Multi-Pattern matching problem:

One of the vital module for a Network based Intrusion detection System is a marching engine whose task is to find the presence of multiple strings or patterns in a given packet payload. In a multi-pattern matching problem, we are given a set of strings S = { s_1, s_2, s_3, ..., s_n } and streaming data T (which is alternatively called text). The objective is to find out all the occurrences of any of the strings in S in T. The strings in S are preprocessed to build a machine. The preprocessing time is not taken into account while designing the engine. Basically, string matching can be considered as a longest prefix matching (LPM) problem. The problem states that is T[i,j] denotes the substring of T starting at location i and ending at location j. If S_l be those set of strings in which each string is of length l bytes, then any l byte string in T staring at location i, i.e. T[i..(i + l + 1)] can match with any of the strings in S_l. So for a given length l, we simply need to look up all the strings in S_l. Now the matching policy is based on decreasing order of the pattern length which means that if a pattern of length l is found in the text, there is no need to search for patterns with length $<$ l.

Related Software and Hardware approaches

Multi-pattern matching is one of the important classical problems in computer science and it is mainly used in NIDS or any network processing applications. There are many varieties of implementations of Network Intrusion detection Systems and these implementations are either software based or hardware based. For every implementation, the basic objective is to reduce the overhead of the string matching module. Software based NIDS use software based string matching algorithms like Rabin-Karp algorithm [35], Knuth-Morris-Pratt(KMP) [36] and Boyer-Moore algorithm [37] which are basically single pattern matching algorithms. Rabin-Karp algorithm uses hashing functions to find a string match whereas Knuth-Morris-Pratt matches the string by comparing character by character. Boyer-Moore which scans the characters form right to left, is usually faster than KMP. Besides these algorithm, another efficient algorithm is Aho-Corasick algorithm [38] which is used

by current version of SNORT. But for high speed network processing applications, some hardware based designs are proposed and they are highly preferred over software based applications. One such approach is use of Finite Automata methods as used in [39]. In such approaches, the signatures or matching patterns are represented using regular expressions and finally they are converted into efficient FPGA based circuit. the main disadvantage in these approaches is that, for a change in the pattern set, the regular expression needs to be recalculated again which is highly complicated. Sourdis [40] proposed a string matching module which was based on pre-coded CAM.. Later Singaraju [41] extended Sourdis's design [40] for fast character matching and achieved a fairly high throughput and resource utilization. Later Dharmapurikar [1] proposed a fast and scalable pattern matching scheme which we have used as the basics of our overall design.

5.3 Proposed Design

In general, an efficient string matching algorithm can be abstracted as a Longest String matching problem [42], which states that in any packet, the strings must be searched in the decreasing order of their length. That means if a pattern of size L is found in a packet at some time instant, then there is no need to search for any other string of size less than L (provided that there is no match found for strings greater than L). Previously, intrusion detections systems were implemented in hardware with various architectures but not all of them were memory efficient. In this section, our proposed architecture for the multi-pattern string matching module is described in details. The algorithm is based on a memory efficient multi hash data structure named Bloom Filter [43] [44] [45]. The speciality of this data structure is that, the computation time involved in performing the query is independent of the number of strings in the database. Moreover a malicious packet cannot escape a bloom filter in any way but sometimes a normal packet can be treated as a malicious packet. Such false positives can be removed by introducing some extra modules. In our work, we have designed a parallel hash module for removing such false positives. The basic idea and design of a bloom

filter is described in subsequent section.

5.3.1 Bloom Filter basics and overview

A bloom filter is a randomized data structure that can represent a set of strings for efficient membership querying. It generally works in two phases. The basic idea is that, given a string S, k hash functions are computed on it, producing k different hash values within range 1 to m. The filter now sets k bits in a m bit long vector at the addresses corresponding to the k hash values. This vector is called as Bloom vector. After this operation, S is made the member of the filter. This procedure is repeated for all the members and this phase is called programming phase of the filter. The next phase for verifying the membership of a string is called the query phase. In this phase, the string under verification is taken as input, and k hash values are generated using the same hash functions of the programming phase. These k values are looked up in the same m bit vector and if any one of these locations is not found set, then that packet is declared as the non member of the set. If all the corresponding bit in those locations are found to be set then the string is said to belong to the set with some probability. Thus there is no chance for presence of false negatives in the output but sometimes false positives may be present. So for these reason, there is a need for a separate analyzer to eliminate those false positives. Figure 5.2 shows a basic architecture of a bloom filter with an analyzer. Thus we can write like this:

- **Programming Phase:**

  $X_i = H_i(S)$ where $0 \leq H_i(a) \leq (m-1)$, X is a set where the hash values are calculated and stored, S is the string to be programmed as a member, m is the size of the Bloom Vector, $0 \leq i < k$, the value of k depends on the programmer.

- **Querying Phase:**

  $Y_i = H_i(S)$ where $0 \leq H_i(a) \leq (m-1)$,Y is the set where the calculated hash values of the string under verification is stored, S is the string under

verification, m is the size of the Bloom Vector, $0 \leq i < k$, the value of k depends on the programmer. If set Y is found to be equal to X then the string S is said to belong to the set with some probability otherwise S is declared as a non member of the set of strings in the filter.

The performance of a bloom filter is given by the following parameters:

- The number of strings to be stored (n).
- The number of associated hash functions for each pattern, denoted by k.
- The size of the Bloom Vector (m).

The probability that a string under test is selected as false positive is given as $(\frac{1}{2})^k$ and the value of m is taken much greater than k.

Now as inputs to a bloom filter are nothing but packet payloads, their size may

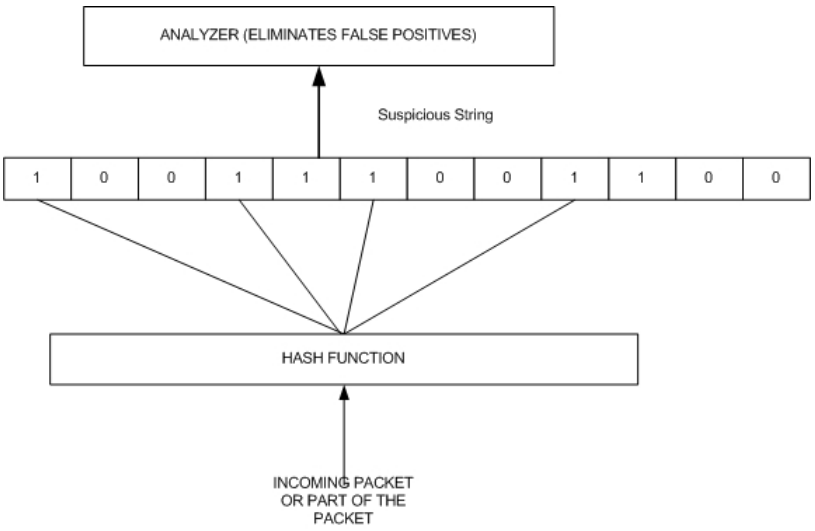

Figure 5.2: A Typical Bloom Filter with an Analyzer

be fixed or variable. Moreover, there may be presence of variable sized multiple patterns on a packet. Now based on the size of the matching patterns, two types of situations may arise. These are discussed below.

5.3.2 Scenario 1: With fixed sized matching patterns:

This situation arises when we know the size and nature of the intrusions beforehand. The string matching module works like a fixed size multiple pattern

matching module. Whenever any packet arises as input, it is scanned for finding any fixed sized patterns within it. The overhead of the matcher is much low as the there is no need to search for patterns with larger or smaller size than the given one. The throughput of the matcher can be increased by replicating the bloom filters, each accepting a fixed sized string as input. The equivalent architecture foe the string matching module is given in Figure 5.3. In the figure, B1, B2 etc are Bloom engines which takes equal sized strings as inputs and work in parallel. Using this architecture, more than one string can be matched in a single clock cycle. For an input packet with a very large payload, a sliding window can be maintained which slides over each and every byte of the string in every clock cycle. For this case, scanning the entire packet may takes more than one clock cycle.

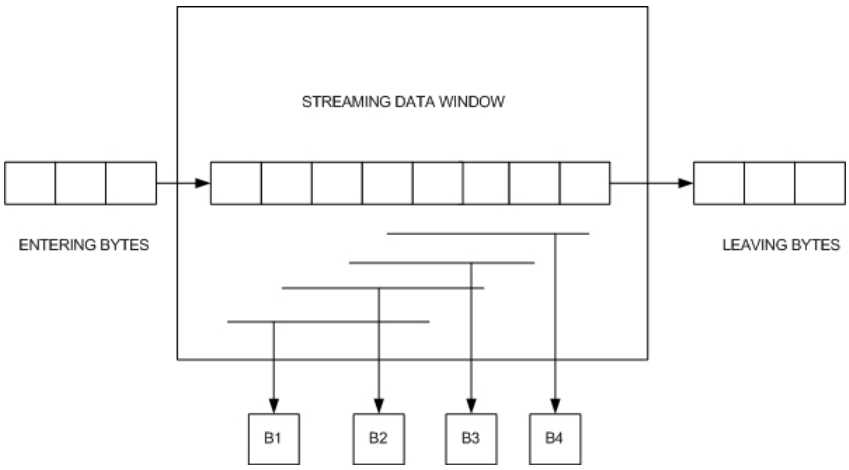

Figure 5.3: Parallel Bloom Filter Matching a fixed sized pattern [1].

5.3.3 Scenario 2: With variable sized matching patterns:

This situation is much more practical because in this case the size of the matching patterns within the packet is not known beforehand. When a packet arrives as an input to the filter, it searches for the patterns of all sizes for which it is programmed. For example, a filter may be programmed with 100 patterns out of which 30 patterns are of size 12 bits, 20 patterns of size 18 bits and the rest of sizes within 10 bits to 6 bits. So in this case, the filter needs to search for all

6,7,8,9,10,12 and 18 bit length patterns in any input string. In this case, it is more efficient to use multiple bloom filters each of which detects a string of unique length, as shown in Figure 5.4. In this case also, more than one string can be matched in a single clock cycle, but here the overhead for the matcher is more as it need for consider all possible lengths of presence of patterns.

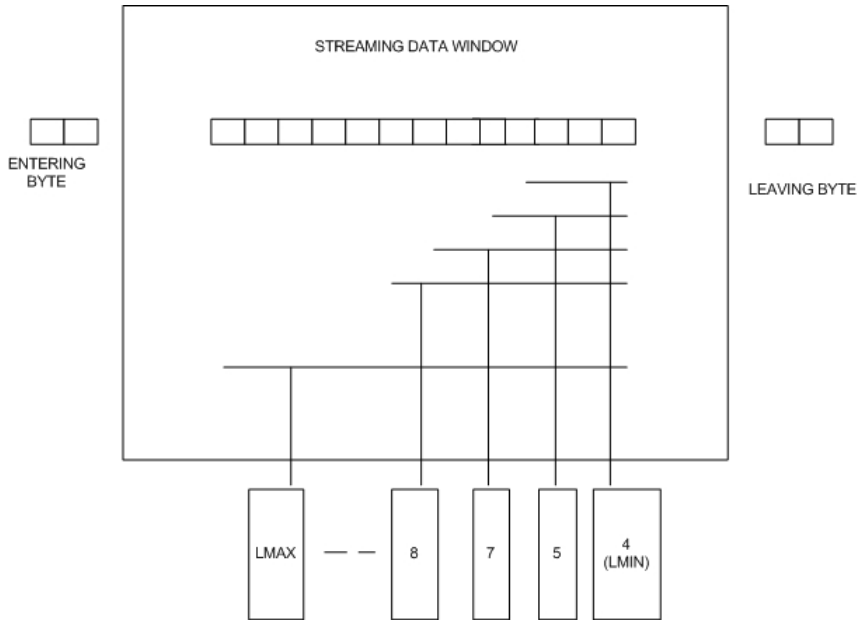

Figure 5.4: A series of Bloom Filters matching variable sized patterns at a time [1].

5.4 Implementation

In this section, we have described our proposed architecture for the multi pattern string matching module for a network based intrusion detection system. Pattern matching for detecting malicious packets involves pattern matching in header as well as the payload of the packet.As the operation is based on deep packet inspection and as the payload scanning is more critical than the header scanning, we only stick to the operation of payload checking. Our assumption is made beforehand that the header has already been scanned for suspicious patterns and the result is negative. We have used Bloom Filter for efficient pattern matching and used a set of rules as sample patterns to match in an input string.

5.4.1 Implementation Constraints

In this work , at first, we have implemented a single bloom filter design for string matching. The design is coded with VHDL and is synthesized using Xilinx-ISE simulator and ChipScope Pro targeting Virtex II pro device. The proposed architecture consists of three modules(shown in Figure 5.6), a partial bloom filter, a hash generator and a module decoder. The algorithm is based on certain assumptions which are:

- The length of the input string is assumed to be fixed(taken as 100 bits in our design).
- The length of the matching patterns is assumed to be fixed(taken as 80 bits in out design).
- The design of the bloom filter is replicated so as to increase the throughput by comparing multiple strings in a single clock cycle. This means a sliding window is chosen and a series of multiple Bloom Filter engines are taken in a cascading orientation, which are capable of matching a fixed size string.

As the size of input string is greater than that of the sliding window, a single byte of the string leaves the window from one side and another new byte enters the window from the other side in every clock cycle. So if the size of the sliding window is increased, the latency of the module reduces gradually. In our implementation, 10 matching patterns of a fixed length(80) are used to program the filter.

5.4.2 Partial and Large Bloom Filter

We have followed the design of [2], which is nothing but the model of a machine problem. To exploit parallelism and flexibility inside the architecture of Bloom Filter as in [46], our main bloom filter architecture is made of small sized modules called Partial Bloom Filter as shown in Figure 5.5. Each of such PBF is capable of comparing 2 hash values generated by the hash functions. The speciality of a partial bloom filter is that apart from the comparison operation, it can modify the bits of the Bloom Vector by means of **bit_data** and **bit_addr** , with **set_bit** as

the control signal. H1 and H2 are the two hash addresses to look up. The control signal **set_bit** is used to check whether the value on **bit_addr** and **bit_data** is valid or not. The whole design is driven by clock and reset control signals as shown in Figure 5.6. Based on the comparison, the output line **partial_bloom_match** gives '1' or '0'. The operation of the PBF is given below.

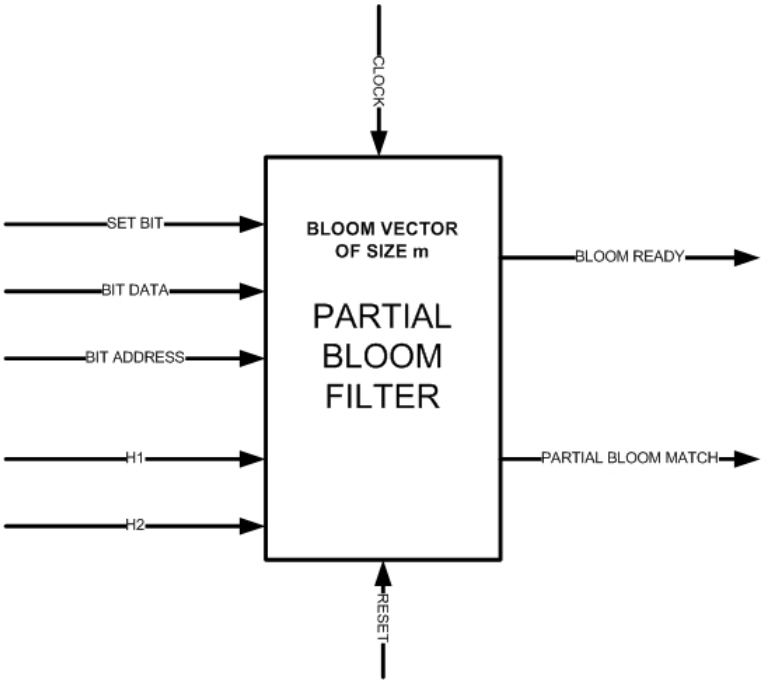

Figure 5.5: Partial Bloom Filter accepting 2 Hash Functions [2]

- **Reset Operation:** When the reset line is high, the Bloom vector is cleared to all '0'. This makes Bloom_ready output as '1'.
- **Configuration:** When the bit_data and bit_addr values are found valid, set_bit becomes '1' and user can update the bits on the Bloom Vector.
- **Querying/Matching:** On each clock cycle, two hash values arrive through H1 and H2 inputs. If the bits located at those values are found '1', then Partial_Bloom_Match outputs '1'.

The large bloom filter is just a collection of partial bloom filter , with a hash generator and a decoder. It has two extra signals along with the common inputs of a PBF which are, valid_request and BRAM number. When the values of BRAM

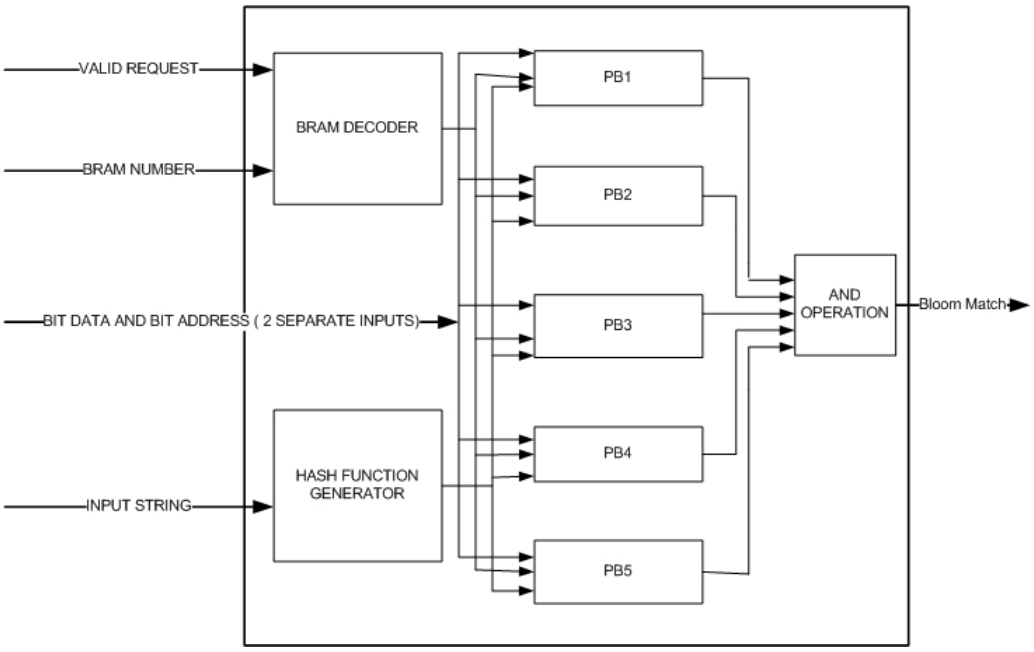

Figure 5.6: Large Bloom Filter using a series of PBFs [2]

number,bit_data and bit_addr are valid, it makes valid_request as '1'. The BRAM number is decoded by the decoder to select one of the PBFs for the operation. In our design, the Hash Function generator computes 10 hash values for an input string and two hash values are fed to each PBF. So a total of 5 PBFs are used as shown in Figure 5.6.

5.4.3 Hash Function

For hardware implementations, there are a separate class of universal hash functions which is proposed in Ramakrishna et.al. [47]. We have used these class of hash functions which are based on random values and Ex-OR operations. In our implementation, we have used 10 different hash functions for each string. By using 10 hash functions, the false positive probability becomes $f = (\frac{1}{2})^{10} = 0.001$. The i^{th} is defined as:

$$H_i(S) = d_{i1}.s_1 \oplus d_{i2}.s_2 \oplus d_{i3}.s_3 \oplus .... \oplus d_{ib}.s_b$$

where

$$S = \{s_1, s_2, ..., s_b\}$$

and the set $\{d_{i1}, d_{i2}, d_{i3}, ..., d_{ib}\}$ is a set of random numbers. In our implementation, this set is taken as a set of random 12 bit values. A table of $1D \times 1D \times 1D$ where the corresponding dimensions are blocks, rows and columns respectively. Each block calculates a separate hash value from the input string. Each row corresponds to one byte of the input string and each column corresponds to a single bit of the input string. The input string is of 80 bit length (10 bytes) and so the dimension of the table is chosen as $10 \times 10 \times 8$. For every byte of the string, the values are calculated in same way as given in [47] (AND and Ex-OR).

5.4.4 Results and Comparison

As mentioned earlier, we have designed the bloom filter using a hash function generator and a series of Partial Bloom Filters (PBF) working in parallel. The input string length is kept fixed for the simulation (80 bits) and it is sub grouped into 10 bytes. Using the $1D \times 1D \times 1D$ table of random numbers, 10 different hash values are generated. The target device is chosen as Virtex II pro XC2VP30 FPGA. The design is coded using VHDL and synthesized using Xilinx ISE. Finally the design is realized in the target FPGA. The synthesis report and the timing summary for the Hash Generator module is shown in Figure 5.7 and Figure 5.8.

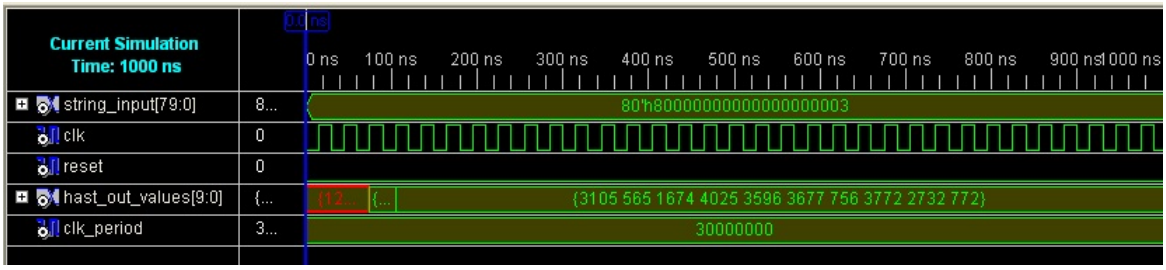

Figure 5.7: Test-bench for the Hash Function Generator

After designing the Hash generator and the equivalent circuit for the Partial Bloom Filter, the Large Bloom Filter (LBF) is designed. From the input string, the Hash Generator generates 10 different Hash Values within the range 0 to $2^{12} - 1$. Each PBF takes 2 of these hash values as input and checks the main Bloom Vector if those positions are set or not. If they are found to be set, the partial bloom match signal goes high. Finally, if all the partial bloom match signals for the partial

```
Timing Summary:
---------------
Speed Grade: -5

   Minimum period: 2.832ns (Maximum Frequency: 353.114MHz)
   Minimum input arrival time before clock: 4.449ns
   Maximum output required time after clock: 3.997ns
   Maximum combinational path delay: No path found

Timing Detail:
--------------
All values displayed in nanoseconds (ns)
```

Figure 5.8: Timing Summary for Hash Generator Module

bloom filters are found to be set, then the bllom match signal goes high. If any one of the partial bloom filter generates a low value for partial bloom match, the bloom match goes low. The matching waveform for the large bloom filter and its timing summary is given shown in Figure 5.9 and 5.10.

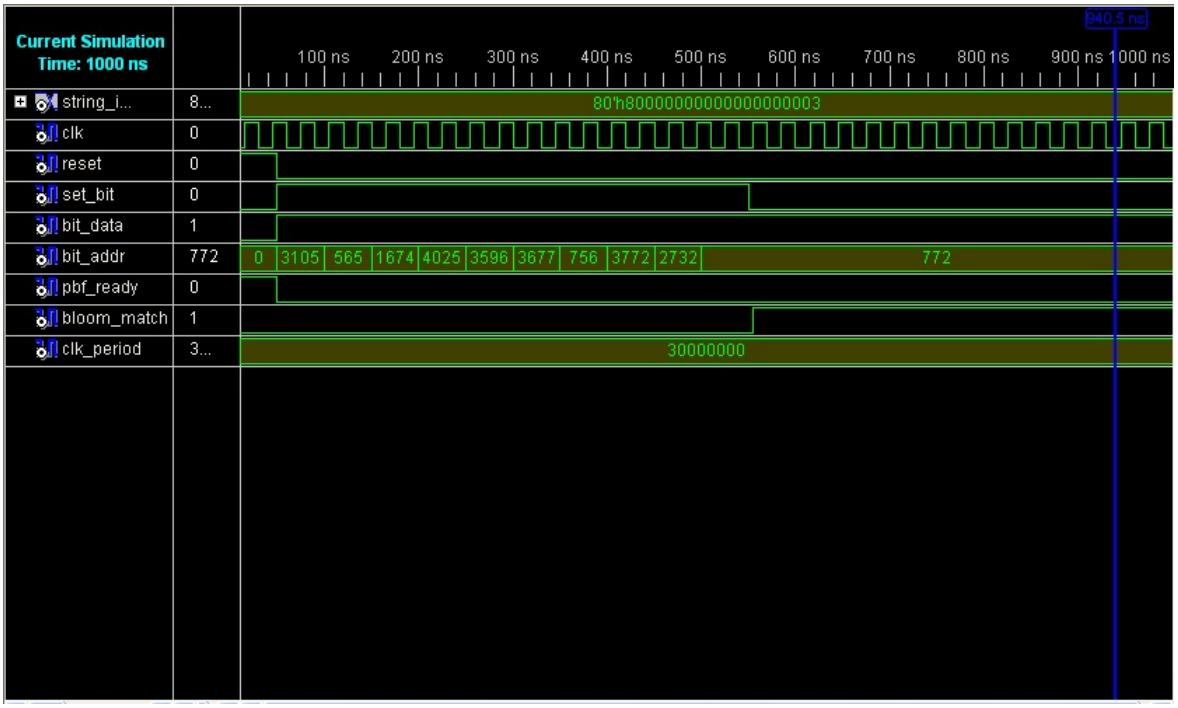

Figure 5.9: Waveform for the overall design for Bloom Filter. Diagram shows the waveform when the supplied Hash values exactly matches with the hash values of the member string, the match signal becomes high

After designing the bloom filter, a scenario is designed where the window size is taken as 80 bit and it is assumed that the size of matching patterns are fixed. The filter is programmed with 10 different patterns and for this design, the throughput

```
Timing Summary:
---------------
Speed Grade: -5

   Minimum period: 9.632ns (Maximum Frequency: 103.823MHz)
   Minimum input arrival time before clock: 9.119ns
   Maximum output required time after clock: 3.997ns
   Maximum combinational path delay: 5.843ns

Timing Detail:
--------------
All values displayed in nanoseconds (ns)
```

Figure 5.10: Timing Summary for the Bloom Filter

if found to be fairly high. From the timing summary of the main filter, the maximum clock frequency is found to be 103.82 MHz and for an input stream of size 80 bits, the throughput is calculated as

$$T = 80 \times 103.82$$

which is equal to 8.11 Gbps. We have compared the performance of our design with two efficient and existing schemes, Singaraju [41] and Sourdis [40], given in Table 5.1.

Table 5.1: Comparison of our proposed design with some existing designs

Design	Device	Throughput	Slices	Input size
Our design	Virtex 2 pro XC2VP30	8.11 Gbps	13125	80 bits
Singaraju's design	Virtex 2 pro XC2V6000	6.41 Gbps	15016	1021 bits
Sourdis's design	Virtex2 1000	12.672 Gbps	18728	32 bits

5.5 Conclusion

In this chapter, we have described a novel technique for multiple pattern string matching algorithm which can be used in a Network Intrusion Detection System. The design is made using a memory efficient multi hashing data structure named Bloom Filter. The speciality of this filter is that, it only allows false positives along with correct matches bit it never allows false negatives. The design is coded using VHDL and synthesized using Xilinx ISE and targeted for Virtex II pro - XC2VP30 FPGA. The maximum operating frequency and the estimated throughput of the design verifies that it can be used as an effective module in a high speed network

to in line speed. In our work, we have not used the internal Block RAMs of the targeting FPGA. In future, we can extend our design considering this fact.

Chapter 6

Conclusion

Conclusion and Future work

Chapter 6

Conclusion and Future Work

In this thesis work, we have discussed about the functionality of network components when implemented in hardware. It is a traditional practice to build any hardware using application specific integrated circuit (ASIC) technology but another issue for such implementation is reconfigurability. So FPGA is the ultimate choice for such practice and out work is just gives the support to this argument. In this thesis, we have proposed some novel pipelined architectures for those computationally intensive modules for network processing applications and they achieved a substantial high throughput. We have chosen a symmetric key block cipher algorithm (IDEA) and a Network Intrusion Detection System architecture for verifying their functionality in FPGA.

However, there has been few constraints associated with our work which we want to sort out in future. Firstly, our work has been realized in Virtex II pro FPGA. In future, we want to verify our design in other High Speed FPGAs. Moreover, we have not used the internal Block RAMs of the FPGA. In future, we may extend our work by using Xilinx Core generator which may reduce the synthesis time. Finally, while designing the Network Intrusion Detection System architecture, we have used certain hash functions for hashing but we have not included a random number generator module for generating the random bits during hashing. In future we will include a new random number generator module in our design. This thesis gives a clear statement that FPGAs are a good candidate for efficient implementation of network processing applications. We hope that there will be some more proposals and implementations of some other algorithms in the near future.

Bibliography

[1] S. Dharmapurikar and J.W. Lockwood. Fast and scalable pattern matching for network intrusion detection systems. *Selected Areas in Communications, IEEE Journal on*, 24(10):1781 –1792, oct. 2006.

[2] Lockwood John W. Acceleration of networking algorithms in reconfigurable hardware, 2003.

[3] Kai Hwang. *Advanced Computer Architecture: Parallelism,Scalability,Programmability.* McGraw-Hill Higher Education, 1st edition, 1992.

[4] Xuejia Lai and James L. Massey. A proposal for a new block encryption standard. In *Proceedings of the workshop on the theory and application of cryptographic techniques on Advances in cryptology*, pages 389–404, New York, NY, USA, 1991. Springer-Verlag New York, Inc.

[5] Xuejia Lai, James L. Massey, and Sean Murphy. Markov ciphers and differential cryptanalysis. In *Advances in Cryptology – CRYPTO '91*, pages 17–38. Springer-Verlag, 1991.

[6] http://en.wikipedia.org/wiki/international_data_encryption_algorithm, 1999.

[7] H. Bonnenberg, Andreas Curiger, Norbert Felber, Hubert Kaeslin, and Xuejia Lai. Vlsi implementation of a new block cipher. In *Proceedings of the 1991 IEEE International Conference on Computer Design on VLSI in Computer & Processors*, ICCD '91, pages 510–513, Washington, DC, USA, 1991. IEEE Computer Society.

[8] A.V. Curiger, H. Bonnenberg, and H. Kaeslin. Regular vlsi architectures for multiplication modulo (2n+1). *Solid-State Circuits, IEEE Journal of*, 26(7):990 –994, July 1991.

[9] A. Curiger, H. Bonnenberg, R. Zimmermann, N. Felber, H. Kaeslin, and W. Fichtner. Vinci: Vlsi implementation of the new secret-key block cipher idea. In *Custom Integrated Circuits Conference, 1993., Proceedings of the IEEE 1993*, pages 15.5.1 –15.5.4, May 1993.

[10] R. Zimmermann, A. Curiger, H. Bonnenberg, H. Kaeslin, N. Felber, and W. Fichtner. A 177 mb/s vlsi implementation of the international data encryption algorithm. *Solid-State Circuits, IEEE Journal of*, 29(3):303 –307, March 1994.

[11] L. Leibowitz. A simplified binary arithmetic for the fermat number transform. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 24(5):356 – 359, October 1976.

[12] Stefan Wolter, Holger Matz, Andreas Schubert, and Rainer Laur. On the vlsi implementation of the international data encryption algorithm idea. In *ISCAS*, pages 397–400, 1995.

[13] S.L.C. Salomao, V.C. Alves, and E.M.C. Filho. Hipcrypto: a high-performance vlsi cryptographic chip. In *ASIC Conference 1998. Proceedings. Eleventh Annual IEEE International*, pages 7–11, September 1998.

[14] O. Mencer, M. Morf, and M.J. Flynn. Hardware software tri-design of encryption for mobile communication units. In *Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on*, volume 5, pages 3045–3048 vol.5, May 1998.

[15] M. P. Leong, Ocean Y. H. Cheung, Kuen Hung Tsoi, and Philip Heng Wai Leong. A bit-serial implementation of the international data encryption algorithm idea. In *FCCM*, pages 122–131, 2000.

[16] Tsoi Kuen Hung. Cryptographic primitives on reconfigurable platforms, 2002.

[17] Seth Copen Goldstein, Herman Schmit, Mihai Budiu, Srihari Cadambi, Matt Moe, R. Reed Taylor, and R. Reed. Piperench: A reconfigurable architecture and compiler. *Computer*, 33:70–77, 2000.

[18] Emeka Mosanya, Christof Teuscher, Héctor Fabio Restrepo, Patrick Galley, and Eduardo Sanchez. Cryptobooster: A reconfigurable and modular cryptographic coprocessor. In *CHES*, pages 246–256, 1999.

[19] Ocean Y. H. Cheung, Kuen Hung Tsoi, Philip Heng Wai Leong, and M. P. Leong. Tradeoffs in parallel and serial implementations of the international data encryption algorithm idea. In *CHES*, number Generators, pages 333–347, 2001.

[20] Antti Hämäläinen, Matti Tommiska, and Jorma Skyttä. 8 gigabits per second implementation of the idea cryptographic algorithm. In *FPL*, pages 760–769, 2002.

[21] Reto Zimmermann. Efficient vlsi implementation of modulo $(2^n + 1)$ addition and multiplication. In *IEEE Symposium on Computer Arithmetic*, pages 158–167, 1999.

[22] Yutai Ma. A simplified architecture for modulo $(2^{\mathrm{n}} + 1)$ multiplication. *IEEE Trans. Computers*, 47(3):333–337, 1998.

[23] Zhongde Wang, Graham A. Jullien, and William C. Miller. An efficient tree architecture for modulo 2+1 multiplication. *VLSI Signal Processing*, 14(3):241–248, 1996.

[24] Ivan Gonzalez, Sergio López-Buedo, Francisco J. Gómez, and Javier Martínez. Using partial reconfiguration in cryptographic applications: An implementation of the idea algorithm. In *FPL*, pages 194–203, 2003.

[25] M. Thaduri, S.-M. Yoo, and R. Gaede. An efficient vlsi implementation of idea encryption algorithm using vhdl. *Microprocessors and Microsystems*, 29(1):1–7, 2005.

[26] Bruce Schneier. *Applied cryptography (2nd ed.): protocols, algorithms, and source code in C*. John Wiley & Sons, Inc., New York, NY, USA, 1995.

[27] L. Leibowitz. A simplified binary arithmetic for the fermat number transform. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 24(5):356–359, oct. 1976.

[28] J.W. Chen and R.H. Yao. Efficient modulo (2^n+1) multipliers for diminished-1 representation. *Circuits, Devices Systems, IET*, 4(4):291–300, jul. 2010.

[29] A. Tyagi. A reduced area scheme for carry-select adders. In *Computer Design: VLSI in Computers and Processors, 1990. ICCD '90. Proceedings., 1990 IEEE International Conference on*, pages 255–258, sep 1990.

[30] Francisco Rodríguez-Henríquez, N. A. Saqib, A. Díaz-Pèrez, and Cetin Kaya Koc. *Cryptographic Algorithms on Reconfigurable Hardware (Signals and Communication Technology)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[31] Pawel Chodowiec, Po Khuon, and Kris Gaj. Fast implementations of secret-key block ciphers using mixed inner- and outer-round pipelining. In *Proceedings of the 2001 ACM/SIGDA ninth international symposium on Field programmable gate arrays*, FPGA '01, pages 94–102, New York, NY, USA, 2001. ACM.

[32] José M. Granado, Miguel A. Vega-Rodríguez, Juan M. Sánchez-Pérez, and Juan A. Gómez-Pulido. Idea and aes, two cryptographic algorithms implemented using partial and dynamic reconfiguration. *Microelectron. J.*, 40:1032–1040, June 2009.

[33] Antti Hamalainen, Matti Tommiska, and Jorma Skytt. 6.78 gigabits per second implementation of the idea cryptographic algorithm. In Manfred Glesner,

Peter Zipf, and Michel Renovell, editors, *Field-Programmable Logic and Applications: Reconfigurable Computing Is Going Mainstream*, volume 2438 of *Lecture Notes in Computer Science*, pages 149–177. Springer Berlin / Heidelberg, 2002. 10.1007/3-540-46117-5-78.

[34] Sarang Dharmapurikar, Praveen Krishnamurthy, Todd Sproull, John Lockwood, and Line Speeds. Deep packet inspection using parallel bloom filters, 2004.

[35] Richard M. Karp and Michael O. Rabin. Efficient randomized pattern-matching algorithms, 1987.

[36] Donald E. Knuth and Vaughan R. Pratt. Fast pattern matching in strings. *Siam Journal on Computing*, 6:323–350, 1977.

[37] Robert S. Boyer and J. Strother Moore. A fast string searching algorithm. *Commun. ACM*, 20:762–772, October 1977.

[38] Alfred V. Aho and Margaret J. Corasick. Efficient string matching: an aid to bibliographic search. *Commun. ACM*, 18:333–340, June 1975.

[39] Reetinder Sidhu and Viktor K. Prasanna. Fast regular expression matching using fpgas. In *in IEEE Symposium on Field-Programmable Custom Computing Machines*, 2001.

[40] Ioannis Sourdis and Dionisios Pnevmatikatos. Fast, large-scale string match for a 10gbps fpga-based network intrusion. *FPL*, 2003:880–889, 2003.

[41] Janardhan Singaraju and John A. Chandy. Fpga based string matching for network processing applications. *Microprocess. Microsyst.*, 32:210–222, June 2008.

[42] Dany Breslauer, Livio Colussi, Laura Toniolo, and Universit'a Di Padova. Tight comparison bounds for the string prefix-matching problem, 1993.

[43] Sarang Dharmapurikar, John Lockwood, and Member Ieee. Fast and scalable pattern matching for network intrusion detection systems. *IEEE Journal on Selected Areas in Communications*, 24:2006, 2006.

[44] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13:422–426, 1970.

[45] Michael Attig, Sarang Dharmapurikar, and John Lockwood. Implementation results of bloom filters for string matching. In *In IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM*, pages 322–323. IEEE Computer Society, 2004.

[46] Bin Xiao and Yu Hua. Using parallel bloom filters for multiattribute representation on network services. *IEEE Trans. Parallel Distrib. Syst.*, 21:20–32, January 2010.

[47] M.V. Ramakrishna, E. Fu, and E. Bahcekapili. A performance study of hashing functions for hardware applications. In *In Proc. of Int. Conf. on Computing and Information*, pages 1621–1636, 1994.

Dissemination of Work

Published

1. Sourav Mukherjee and Bibhudatta Sahoo, ‟A novel modulo $(2^n + 1)$ multiplication approach for IDEA cipher″, Published in *International Journal of Programmable Device Circuits and Systems*, pp 187 - 193, Volume 2, Number 11, November, 2010.

2. Sourav Mukherjee and Bibhudatta Sahoo, ‟A Hardware implementation of IDEA cryptosystem using a recursive multiplication approach″, *International Conference on Electronic Systems (ICES-2011)*, pp 383 - 389, 2011, New Delhi, India.

Accepted

1. Sourav Mukherjee and Bibhudatta Sahoo, ‟A Survey on Hardware implementation of IDEA Cryptosystem″, *Information Security Journal: A Global Prospective*, Volume 20, Number 6, February, 2011.

Communicated

1. Sourav Mukherjee and Bibhudatta Sahoo, ‟An Improved IDEA implementation on FPGA using an efficient diminished-one modulo multiplication based on Radix 8 Booth's recoding″, *Journal of Cryptographic Engineering (SPRINGER)*, Volume 1, 2011.